Mendelova univerzita v Brně
Provozně ekonomická fakulta

Refaktoring portálu Curriculum Generator na single-page aplikaci

Diplomová práce

Vedoucí práce:

Ing. Ondřej Popelka, Ph.D.

Bc. Jakub Stratil

Brno 2017

Děkuji vedoucímu práce Ing. Ondřeji Popelkovi, Ph.D. za vstřícnost, odborné vedení a cenné rady při vypracovávání této diplomové práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci:
Refaktoring portálu Curriculum Generator na single-page aplikaci
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 22. května 2017 _____

Abstract

Stratil J. Refactoring of web portal Curriculum Generator to single-page application. Diploma thesis. Brno: Mendel University, 2017.
This thesis is documenting the refactoring process of web application located at web portal Curriculum Generator. The thesis contains analysis of the application's status, which is the basis for a draft. The draft includes changes to be made during the refactoring process, a process of choosing JavaScript framework for the refactoring process and a draft of new API. The thesis continues with an implementation of the proposed changes and draft, using chosen technologies. The thesis ends with a description of the refactored application's deployment process.

Keywords

Web application, Single-page application, REST API, JavaScript framework, PHP framework, Angular, Lumen, JSON API, JSON Web Token, JWT

Abstrakt

Stratil, J. Refaktoring portálu Curriculum Generator na single-page aplikaci. Diplomová práce. Brno: Mendelova univerzita v Brně, 2017.
Práce dokumentuje proces refaktoringu webové aplikace nacházející se na portálu Curriculum Generator. Práce obsahuje analýzu současného stavu aplikace, jenž je základem pro návrh. Návrh zahrnuje změny prováděné v rámci refaktoringu, výběr JavaScriptového frameworku pro refaktoring a návrh API. Následně je popsána implementace vypracovaných návrhů pomocí vybraných technologii. Práce je zakončena popisem procesu nasazení refaktorované aplikace do produkčního prostředí.

Klíčová slova

Webová aplikace, Single-page aplikace, REST API, JavaScriptový framework, PHP framework, Angular, Lumen, JSON API, JSON Web Token, JWT

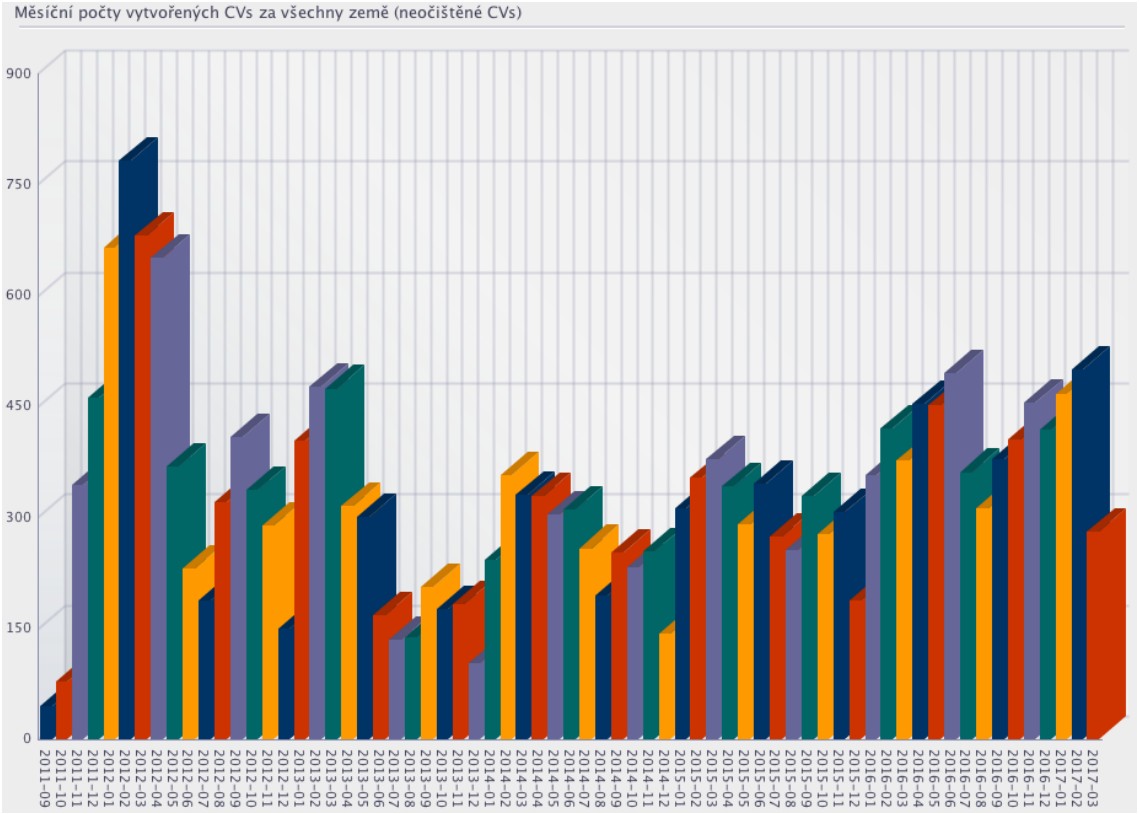
Obsah

1 Úvod

Pro firmu Nobel System pracuji od podzimu roku 2014, kdy jsem začal pracovat na projektu, na kterém pracuji doteď. Tato firma spravuje několik dalších webových projektů a jedním z těchto projektů je webová aplikace Curriculum Generator.

Hlavním účelem webové aplikace Curriculum Generator, jenž se nachází na webové adrese www.curriculum-generator.com, je vytváření strukturovaného životopisu prostřednictvím formuláře. Z vytvořeného životopisu lze následně vygenerovat PDF, ale pro vygenerování souboru, s kterým lze volně nakládat, musí uživatel zaplatit.

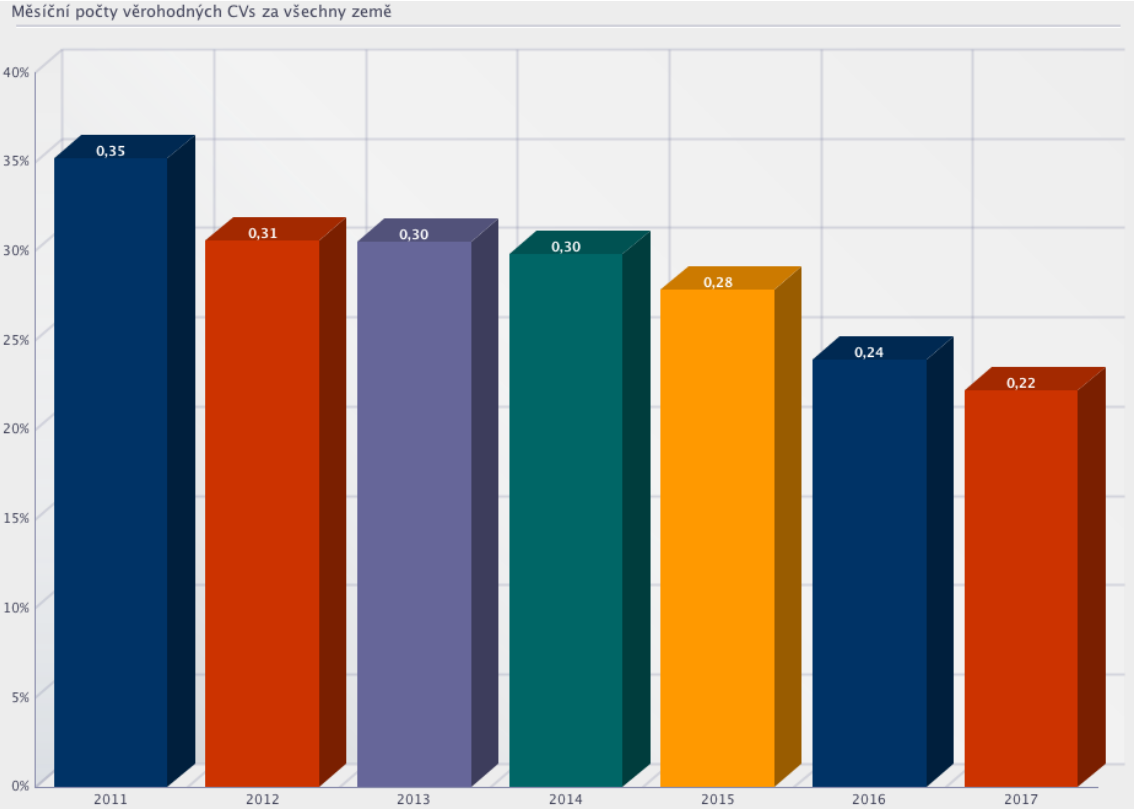
Aplikace je v provozu již od roku 2011 a od té doby nebyly v aplikaci provedeny žádné výrazné změny (viz kapitola 3). Tento fakt, ve spojení s větší konkurencí, se projevuje v tom, že ačkoliv se počet vytvářených životopisů od roku 2012 výrazně neměnil (viz obr. 1), stále méně uživatelů životopis skutečně smysluplně vyplní (viz obr. 2) a o to méně uživatelů za životopis také zaplatí.



Obr. 1 Počet životopisů vytvořených v jednotlivých měsících od spuštění aplikace.

Firma rozhodla, že se refaktoringem aplikace pokusí situaci zlepšit. Refaktorovaná aplikace by měla být optimalizována pro použití nejen na počítačích, ale také na mo-

bilních zařízeních. Firma proto určila, že by bylo vhodné refaktoring provést s využitím nejnovějších technologii, pomocí kterých lze vytvořit aplikaci s vysokou úrovní interakce a uživatelské přívětivosti.



Obr. 2 Procentuální poměr počtu věrohodných životopisů oproti vytvořeným.

2 Cíl práce

Cílem této práce je refaktorování současné webové aplikace Curriculum Generator na single-page aplikaci. Prvním krokem pro dosažení tohoto cíle je analyzování současného stavu aplikace. Důležité je analyzovat především současné uživatelské rozhraní a zhodnotit možnosti použití aplikace na mobilních zařízeních. Dále by se měl analyzovat současný datový model a v neposlední řadě také architektura aplikace. Tam je vhodné se zaměřit na v současnosti použité technologie aplikace a možnost vytvoření single-page aplikace.

Na základě poznatků získaných analýzou současného stavu, je potřeba vytvořit návrh změn uživatelského rozhraní a nové single-page aplikace tak, aby byla použitelná i na mobilním zařízení. Pro single-page aplikaci je potřeba vybrat vhodný JavaScriptový framework a následně navrhnout REST rozhraní schopné předávat data ze současného datového modelu do nově vytvořené single-page aplikace.

Posledním krokem je samotná implementace navrhnuté aplikace s využitím vhodné technologie pro REST rozhraní a použitím vybraného JavaScriptového frameworku k vytvoření single-page aplikace.

3 Současný stav aplikace

Současná webová aplikace se nachází na webové adrese www.curriculum-generator.com. Hlavním účelem této aplikace je tvorba strukturovaného životopisu, který lze následně vygenerovat ve formátu PDF. Výsledný soubor tedy nelze upravovat jinak než pomocí aplikace Curriculum Generator.

3.1 Uživatelské rozhraní

Webová aplikace byla poprvé spuštěna v roce 2011 a od té doby nebyla nijak výrazně upravována, což ilustrují např. obr. 3, na kterém lze vidět současnou úvodní stránku a obr. 4, na kterém je zachyceno, jak stránka vypadala v říjnu 2011.

Obr. 3 Úvodní stránka webové aplikace
Zdroj: Curriculum-Generator.com, 2016

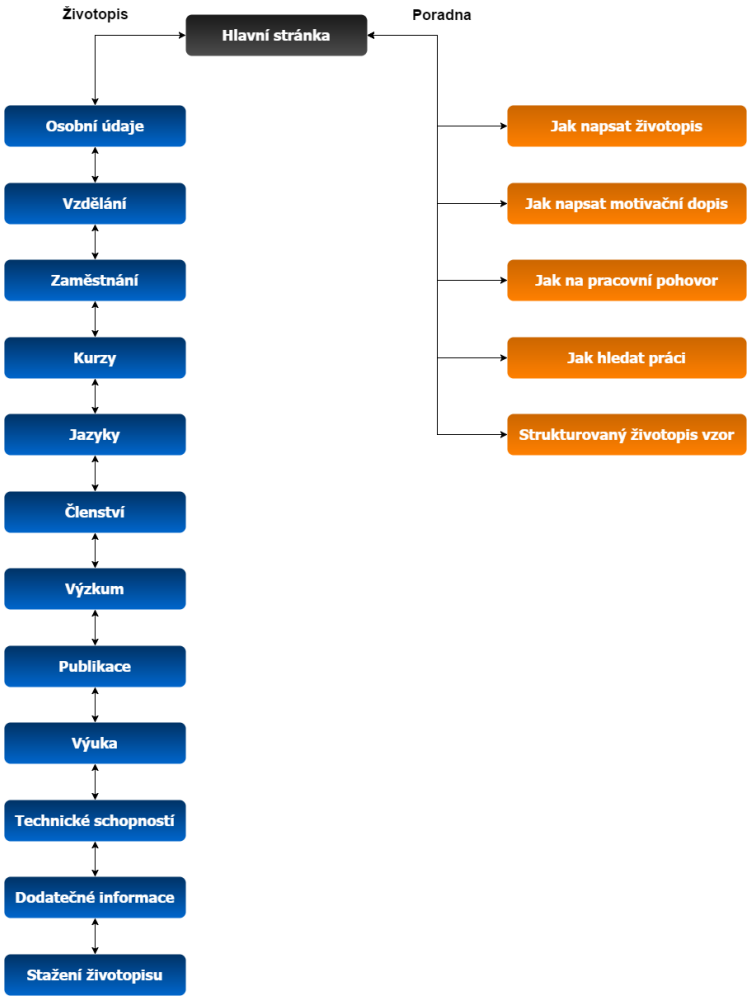
Rozdíly jsou tedy minimální, změnily se pouze některé texty, přibyly jazykové varianty a některé odkazy na sociální sítě. Výrazně se neliší ani další části aplikace.

3.1.1 Části aplikace

Současná aplikace se skládá ze tří hlavních obsahových celků (viz obr. 5):

1. Úvodní stránky – přístupový bod a zároveň rozcestník.
2. Poradny – obsahuje návody Jak napsat životopis, Jak napsat motivační dopis, Jak na pracovní pohovor, Jak hledat práci a Strukturovaný životopis vzor. Horní polovina hlavní stránky je i zde.
3. Formuláře pro tvorbu životopisu.

Obr. 4 Úvodní stránka webové aplikace v roce 2011.
Zdroj: Archive.org, 2017



Obr. 5 Architektura současné aplikace Curriculum Generator

3.1.2 Tvorba životopisu

Uživatel zadává údaje pro výsledný životopis prostřednictvím formuláře (viz obr. 6) na 9-12 krocích (v závislosti na zvoleném typu životopisu) (Curriculum-Generator.com, 2016):

1. Osobní údaje[1]
2. Vzdělání
3. Zaměstnání
4. Kurzy
5. Jazyky
6. Členství
7. Výzkum[2]
8. Publikace[4]
9. Výuka[4]
10. Technické schopnosti
11. Dodatečné informace
12. Stažení životopisu

Obr. 6 První krok formuláře – Osobní údaje
Zdroj: Curriculum-Generator.com, 2016

Vyplněný životopis lze následně vygenerovat ve dvou variantách (Curriculum-Generator.com, 2016):

1. Zdarma, ale v pozadí se nachází logo stránky jako vodoznak a životopis nelze vytisknout nebo z něj kopírovat.
2. Za poplatek, bez vodoznaku a bez jakýchkoliv omezení souboru.

[1] Po odeslání prvního formuláře aplikace uživateli automaticky vytvoří účet a vygeneruje náhodné heslo, které zašle na poskytnutý e-mail. Následně aplikace uživatele také automaticky přihlásí.
[2] Vynecháno u životopisu typu *Absolvent*

3.1.3 Nedostatky uživatelského rozhraní

Jak již bylo zmíněno, aplikace se od roku 2011 nijak výrazně neměnila a tento fakt se odráží na současném stavu aplikace. Aplikace je sice funkční a plní svůj účel, ale zároveň stav aplikace nelze nazvat optimálním. Mezi hlavní chyby a nedostatky patří např.:

- Typy životopisu *Vyučený* a *Manažer* vedou na zcela totožný formulář o 12 krocích, ačkoliv první krok u typu životopisu *Vyučený* ukazuje, že pro tento typ životopisu by kroků mělo být jen 8.
- Mezi jednotlivými kroky formuláře se lze přesouvat jen po 1 kroku logikou předchozí nebo další krok. Pro zobrazení náhledu životopisu se tak musí projít všemi kroky.
- Každý účet může současně spravovat pouze jeden životopis.
- Neoptimalizovaný responzivní design. Stránka se s menší velikostí rozlišení pouze úměrně zmenšuje, a ačkoliv si stránka zachovává použitelnost, ovládací prvky již nejsou dále optimalizovány, viz obr. 7 a obr. 8.

  Tento stav není v dnešní době ideální, protože značná část uživatelů používá zařízení s rozlišením obrazovky menším jak 1366x768px (GS.StatCounter.com, 2017).

Obr. 7 Hlavní stránka při rozlišení o šířce 768px.
Zdroj: Curriculum-Generator.com, 2016

Obr. 8 Ovládací prvky formuláře při rozlišení o šířce 768px.
Zdroj: Curriculum-Generator.com, 2016

3.2 Datový model

Informace, které uživatelé zadají prostřednictvím uživatelského rozhraní (neboli údaje pro vytvoření životopisu) jsou ukládány do relační databáze používající systém MySQL. Současná databáze obsahuje 27 tabulek (viz schéma na obr. 9 a příloha a) a kromě dat získaných od uživatelů obsahuje také několik tabulek s daty technického rázu (log akcí uživatelů, texty návodů v poradně, aj.)

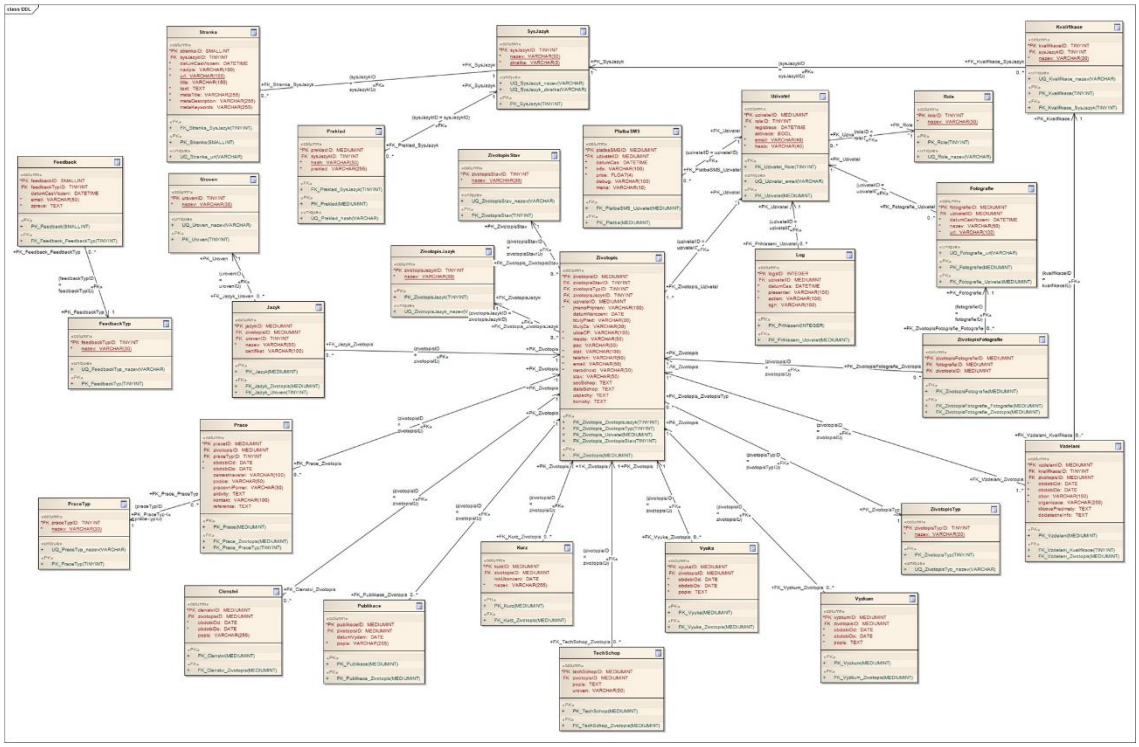
Ústřední tabulkou modelu je tabulka *Zivotopis*, na kterou se dále napojuje 14 tabulek vztahem typu 0:N nebo 1:N, ať už v jednom či druhém směru. Např. jeden uživatel může mít více[3] životopisů a jeden životopis může obsahovat více vzdělání.

3.2.1 Nedostatky datového modelu

Datový model nemá žádné kritické chyby nebo nedostatky, ale vzhledem k tomu, že některé tabulky neobsahují žádná data a některé další lze převést na sloupce (více viz podkapitola 4.4), lze současný datový model považovat za naddimenzovaný. Databáze byla nejspíše připravena pro případná rozšíření, která se neuskutečnily nebo nebyly v předpokládaném rozsahu.

Ideální není také české pojmenování názvů tabulek a sloupců. To ale nezpůsobilo žádné komplikace, protože aplikaci vytvářel a po celou dobu spravoval pouze český tým lidí.

[3] To současná aplikace ovšem reálně nepodporuje.



Obr. 9 Schéma databázového modelu

3.3 Implementace aplikace

Aplikace je implementována jako jednotný celek, jež je jádrem celé webové aplikace. Zahrnuje jak veškerou logiku, výpočty a funkčnost, tak i strukturu a vzhled uživatelského rozhraní, jehož prostřednictvím uživatelé zadávají vstupní data, které aplikace validuje a dále zpracovává.

3.3.1 Architektura aplikace

Základem implementace je původem český tzv. „full-stack“[4] PHP framework Nette, konkrétně pak ve verzi 2.0.0-dev[5] ze srpna 2011. Nette je postaveno na architektuře Model-View-Controller (MVC). (Nette.org, 2017)

Použití Nette bylo, s ohledem na požadavky a velikost aplikace, vhodnou volbou. Již v použité verzi 2.0.0 obsahovalo většinu funkcionality potřebné k vytvoření cílové aplikace (Nette.org, 2012):

- Vlastní databázovou vrstvu
- Vlastní systém šablon – Latte

[4] Full-stack framework pomáhá při vývoji všech částí ucelené aplikace. Od práce s datovým úložištěm až po vytváření uživatelského rozhraní. (Blog.AppDynamics.com, 2015)

[5] Dev neboli development označuje verzi, která se stále vyvíjí. V tomto případě byla finální verze 2.0.0 vydána až na začátku února 2012 (Nette.org, 2012).

- Vlastní ladící nástroj
- Vlastní systém formulářů a požadavků
- Cachování
- Dependency Injection
- Posílání e-mailů
- Zabezpečení

Jak je z předchozího výčtu patrné, Nette splňuje skoro všechny funkční požadavky pro tvorbu této aplikace pomocí PHP. Jedním z požadavků, pro nějž Nette nemá vlastní řešení, je generování PDF souborů. K tomuto účelu byla tedy využita PHP knihovna mPDF (mPDF.GitHub.io, 2015).

3.3.2 Nedostatky aplikační vrstvy

Jak bylo uvedeno na začátku podkapitoly, uživatelské rozhraní úzce souvisí s implementací. Z toho důvodu jsou zmíněné nedostatky uživatelského rozhraní způsobeny více či méně také právě nedostatečnou nebo chybnou implementací.

Na dalších případných nedostatcích implementace má svůj podíl především technologický pokrok. Tato aplikace byla vytvořena ve své době standardním postupem s využitím zaběhnutých technologii a v rámci možností by se aplikace mohla modernizovat při zachování stávající struktury i v současnosti.

V roce 2011 (nebo i později) se teprve začaly vyvíjet a používat technologie, které v dnešní době umožňují efektivnější přístup ke zpracování tohoto typu aplikací (více viz kapitola 4).

4 Návrh

Z nedostatků jednotlivých částí, objevených při analýze současného stavu aplikace, vyplývá potřeba některých oprav a zlepšení. Hlavní změny, které je potřeba udělat, a jež se zároveň stávají cíli samotného refaktoringu aplikace, jsou:

1. Přepracování hlavní stránky.
   1.1. Představení funkce a obsahu aplikace uživateli.

   Hlavní stránka by měla i nadále uživateli zřetelně nabízet funkci vytvoření životopisu a ukázat, jak výsledný životopis může vypadat. To je totiž pro většinu uživatelů důležitým aspektem při rozhodování, jestli aplikaci použít.

   Na závěr je vhodné dát uživateli vědět, že v aplikaci nalezne také další informace související s hledáním a získáváním práce a nemusí tento druh informací nutně hledat jinde.

   1.2. Použití minimálního množství informací a zachování dobré přehlednosti.

   Současná hlavní stránka má relativně hodně prvků (textů, tlačítek atd.) na malém prostoru. Cílem bude hlavní stránku odlehčit, informace více rozprostřít a některé odstranit.

2. Přepracování uživatelského procesu vytváření životopisu.
   2.1. Sjednocení 3 typů životopisu do jednoho.

   Současné rozdělení životopisu na 3 typy (*Vyučený*, *Absolvent*, *Manažer*) neplní žádný podstatný účel. Navíc může být pro uživatele matoucí a nasměruje ho k volbě typu, který nakonec ani nebude vyhovovat jeho potřebám.

   Jak bylo také zmíněno v předchozí kapitole, volba konkrétního typu momentálně nefunguje korektně, což jen potvrzuje, že možnost volby postrádá smysl.

   2.2. Poskytnutí možnosti přímé volby konkrétního kroku.

   Samotný systém formulářů, použitý k zadání údajů pro tvorbu životopisu, není špatný a může být zachován ve skoro neměnné podobě. Ovšem omezení, kdy se lze současně přesunout pouze o jeden krok, je velkým nedostatkem. V tomto ohledu je potřeba uživatelské rozhraní rozšířit o ovládací prvek, např. nějaký druh nabídky, poskytující možnost vybrat libovolný krok. Toto vylepšení by mělo být velkým přínosem pro potřeby uživatele.

   2.3. Vylepšení – permanentní náhled životopisu.

   Není žádoucí, aby uživatel musel generovat nový PDF soubor s životopisem pokaždé, když chce vidět, jak životopis vypadá po vyplnění určitých informací nebo provedení změn.

Nabízí se proto řešení, jehož cílem je rozšířit uživatelské rozhraní tvorby životopisu o náhled, který bude uživateli poskytovat okamžitou odezvu a v rámci možností reprezentovat výsledný dokument.

3. Rozšíření možností uživatelského účtu.

   Jasným cílem tohoto rozšíření je přidání více možností, jak může uživatel svůj životopis spravovat, nebo nově, jak bude uživatel moci mít a spravovat svých životopisů několik. Důležitá může být pro uživatele např. možnost vymazání životopisu, ať už z jakýchkoliv důvodů. Díky této změně lze také přidat nějaké způsoby nastavení účtu, první potenciální funkcí by mohla být např. změna hesla.

4. Modernizovat responzivitu a celkový vzhled aplikace.

   Současný vzhled aplikace, ačkoliv stále použitelný, již působí zastaralým dojmem. Jedním z cílů je tedy celková modernizace designu aplikace s ohledem na požadavek responzivity. S tím souvisí také designové změny, které jsou potřeba v rámci 3. bodu tohoto výčtu.

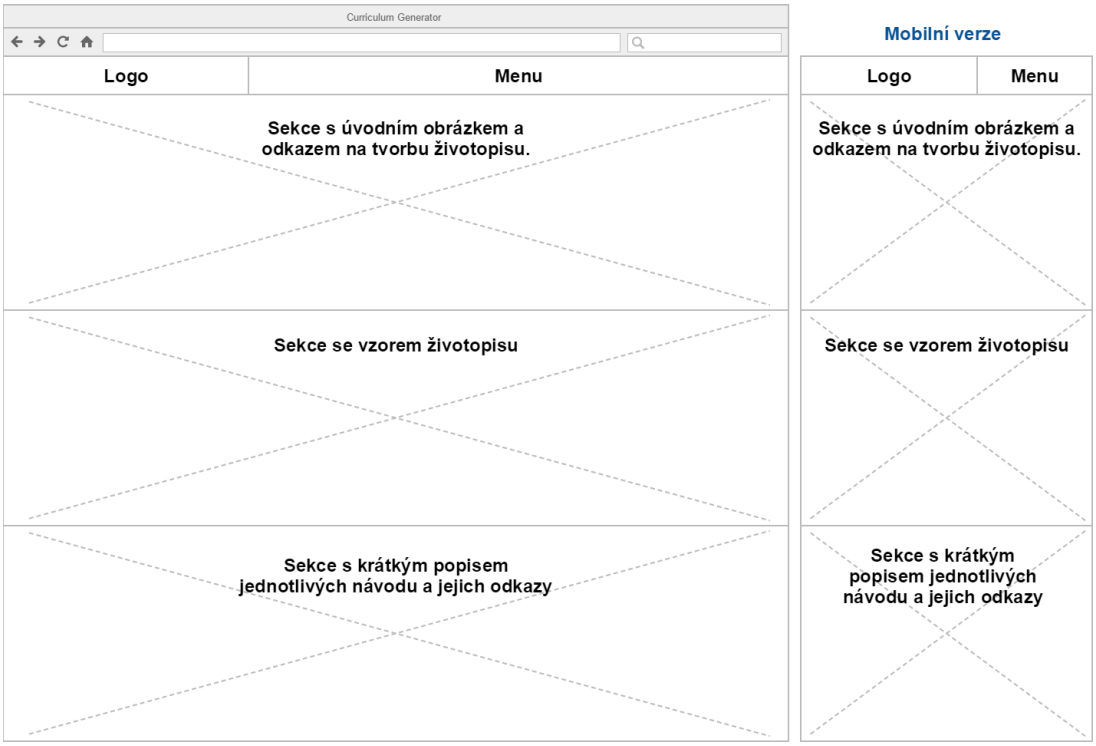
5. Úprava databáze tak, aby více odpovídala skutečným potřebám aplikace.

   Snaha zjednodušit databázi s přihlédnutím k nedostatkům zmíněných v předchozí kapitole. Případné další změny databáze se pravděpodobně také budou odvíjet podle ostatních prováděných změn.

4.1 Drátěné modely

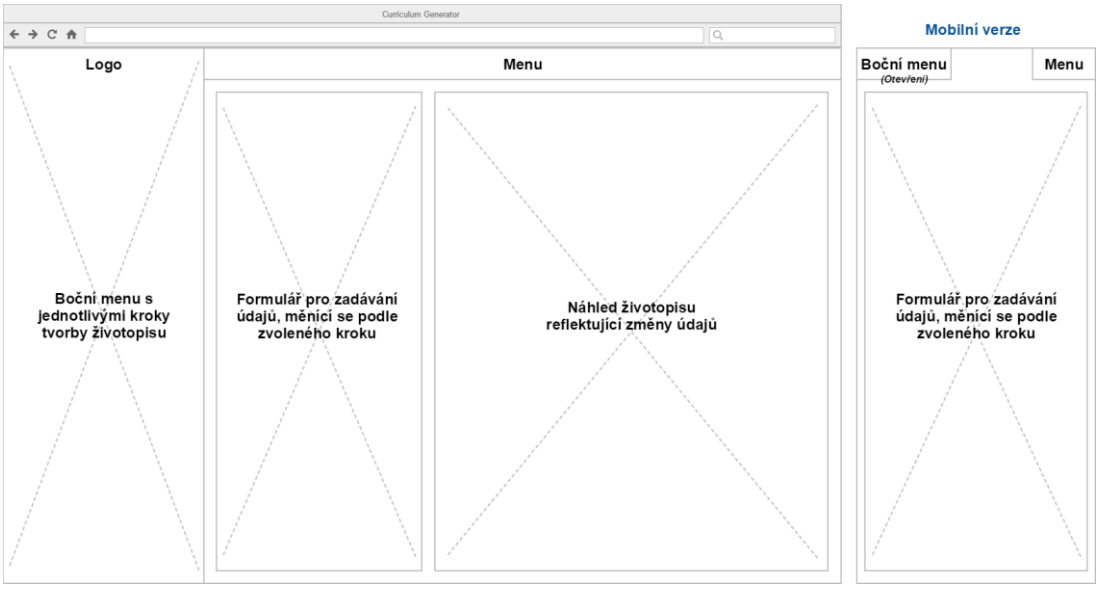
Podle navrhnutých změn byly sestaveny drátěné modely zobrazující návrh možného nového rozložení aplikace, které splňuje všechny stanovené požadavky. Využívá celou šířku okna prohlížeče a vedlejší boční menu s dalšími akcemi v závislosti na aktuální stránce. Aplikace si bude udržovat základní rozložení, dokud to šířka rozlišení dovolí. Na zařízeních s menším rozlišením bude boční menu (případně i náhled) ve výchozím stavu skryté a půjde otevřít tlačítkem v levém horním rohu stránky.

První model znázorňuje hlavní stránku (viz obr. 10), která je rozdělena do 3 obsahových sekcí: úvodní sekce odkazující na tvorbu životopisu, sekce s náhledem životopisu a sekce s krátkým popisem jednotlivých životopisů a jejich odkazy.



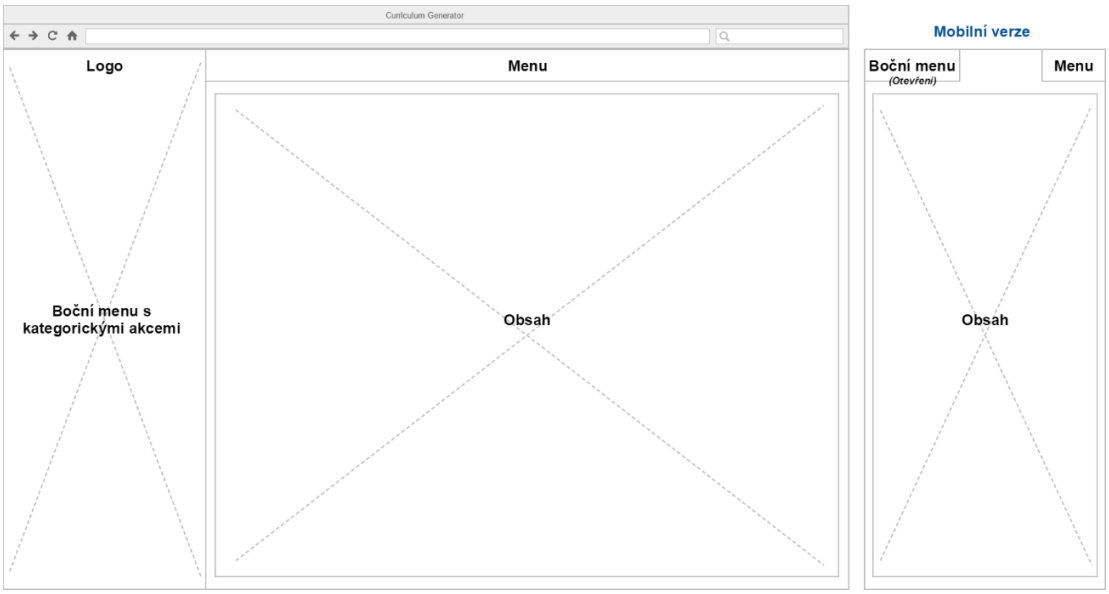
Obr. 10 Drátěný model hlavní stránky.

Druhý model znázorňuje stránku pro tvorbu životopisu rozšířenou o náhled životopisu (viz obr. 11). Využívá boční menu pro snadné přepínání mezi jednotlivými kroky.



Obr. 11 Drátěný model stránky pro tvorbu životopisu.

Poslední model znázorňuje stránku s návody a stránku uživatelského účtu (viz obr. 12). Tyto stránky obsahují boční menu s kategorickými akcemi a zbylý prostor uživatelského rozhraní je určen obsahu.



Obr. 12 Drátěný model stránky s návody a stránky uživatelského účtu.

4.2 Výběr JavaScriptového frameworku

Navrhnuté cíle a změny kladou určité požadavky na technologii, která bude použita k refaktorování současné aplikace. Při tvorbě životopisu se předpokládá vyplňování údajů alespoň v některých krocích, ne-li ve všech. Přitom je potřeba udržovat všechny údaje zadané v jednotlivých krocích pro další manipulaci a zobrazení v náhledu životopisu. Ten by měl navíc okamžitě reflektovat každou změnu, kterou uživatel provede.

V současné aplikaci lze zmíněnou interakci udělat jen velice komplikovaně. Bylo by nutné použít JavaScript (JS) a při každé změně kroku by se musely načíst všechny údaje životopisu, aby byly dostupné pro náhled. Uživatel by navíc ztrácel momentálně vyplněné údaje nebo by se ho aplikace musela neustále dotazovat, jestli chce údaje uložit.

Nabízí se proto řešení pomocí single-page aplikace (SPA). SPA jsou webové aplikace, které pracují pouze s jednou webovou stránkou, která je dynamicky upravována a aktualizována v reakci na úkony uživatele, přičemž není znovu načítána. SPA se tak ve výsledku může podobat nativní aplikaci. (Mikowski, Powell, 2012, s. 1-4)

Přepracování současné aplikace na SPA lze realizovat např. pomocí některého JavaScriptového (JS) frameworku. JS frameworky často využívají ve své struktuře návrhový vzor MVC, vlastní alternativu k tomuto vzoru nebo se zaměřují jen na některou z vrstev. JS frameworky seskupují užitečné komponenty, další knihovny

apod. do jednoho celku, který je tak rámcem pro efektivnější vývoj webových aplikací pomocí JS. (Keefer, 2015)

JS frameworky, které patří mezi nejlépe hodnocené (GitHub.com, 2017):

- AngularJS/Angular
- React
- Vue.js
- Ember.js

Refaktorovaná aplikace má jediný účel (tvorbu životopisů) a přidávání nových funkcí se neplánuje. Navíc je vhodné ji naimplementovat co nejrychleji a s nejsnazší možnou údržbou. Zvolený JS framework by tedy měl být, pokud možno, jedinou technologii potřebnou k vytvoření uživatelského rozhraní a jeho funkční logiky. Nutností jsou také vlastnosti a funkce potřebné pro zpracování cílů definovaných na začátku této kapitoly. Hlavní požadované vlastnosti a funkce:

- Dependency Injection (DI).
- Zpracování a validace formuláře.
- Routování.
- Posílání a přijímání dat neboli schopnost práce s HTTP požadavky.
- Komponentový přístup.
- Možnost rozšíření o externí knihovny.

Pomocí těchto požadavků byly vyhodnoceny a testovány výše vypsané JS frameworky.

4.2.1 AngularJS

AngularJS (Angular 1) je JS framework vyvíjený a spravovaný firmou Google od roku 2010 (AngularJS.org, 2010). AngularJS je určen především pro dynamické webové CRUD[6] aplikace a používá klasické HTML šablony a JavaScript, které rozšiřuje o novou syntax – direktivy. AngularJS je také volně modifikovatelný a kompatibilní s dalšími knihovnami. (AngularJS.org, 2017)

Lze tedy využívat např. (AngularJS.org, 2017):

- Deklarativní programování[7]
- Znovupoužitelných komponent
- Datové vazby
- Vkládání závislostí
- Routování

[6] CRUD značí čtyři základní operace používané při práci s daty – Create, Read, Update, Delete.

[7] Specifikuje se, co se má udělat, ale už ne jak – o to se v tomto případě stará framework.

4.2.2 Testování AngularJS

AngularJS (verze 1.5) byl otestován prostřednictvím tutoriálu, který se nachází na oficiálních stránkách (AngularJS.org, 2017). V rámci tutoriálu se vytváří SPA, jejímž hlavním obsahem je seznam mobilních telefonů (viz obr. 13) implementující UI (User Interface) vzor master-detail[8]. Tutoriál obsahuje dostatečné množství informací, které předává srozumitelnou formou. Důležitá témata, kterým se tutoriál mj. věnuje, jsou např.:

- Vytváření komponent.
- Použití dvousměrných datových vazeb a direktiv v šablonách.
- Využití DI.
- Tvorba routování.
- Práce s HTTP požadavky.
- Animacím.

Formuláře, které tutoriál moc nevyužívá, jsou podrobně popsány v dokumentaci na webových stránkách frameworku (AngularJS.org, 2017). Výsledkem tedy je, že framework splňuje všechny stanovená kritéria a tím pádem by šel použit k refaktorování současné aplikace na SPA.

Obr. 13 AngularJS aplikace PhoneCat – seznam zařízení
Zdroj: AngularJS.org, 2017

4.2.3 Angular

Angular (Angular 2) se začal vyvíjet v roce 2014 a v srpnu 2016 byl oficiálně vydán jako samostatný (a nekompatibilní) nástupce AngularJS (Angular.io, 2016). Angular volně navazuje na svého předchůdce, jehož funkcionalitu podstatně mění a částečně rozšiřuje (Angular.io, 2017):

- Má modulární architekturu, díky níž je lépe strukturovaný a více kompatibilní s externími knihovnami.

[8] V uživatelském rozhraní master-detail je zobrazen seznam objektů, kde uživatel výběrem jednoho z objektů zobrazí detail (podrobnější informace a možnosti) pro zvolený objekt. (Angular.io, 2017)

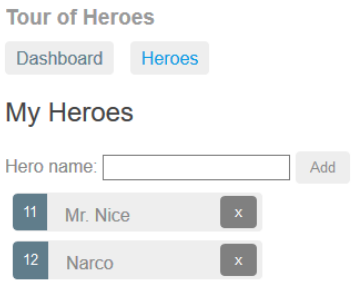
- Podporuje vývoj aplikací pro různé platformy, ať už nativní (desktopové, mobilní) nebo multiplatformní (webové).
- Zaměřuje se na maximální možnou výkonost a škálovatelnost.
- Primárním jazykem je TypeScript[9] (TS), který je doporučený, ale lze používat i JS, do kterého se TS kompiluje.

4.2.4 Testování Angularu

Angular (verze 2.4) byl otestován prostřednictvím TS verze oficiálního tutoriálu (Angular.io, 2017). Pomocí tutoriálu se vytváří SPA, jejíž základ opět tvoří UI vzor master-detail. Výsledná aplikace – seznam hrdinů (viz obr. 14), představuje jednoduchou, ale plnohodnotnou CRUD aplikaci, kde se dají jednotlivé záznamy také přidávat, mazat a měnit.

Ačkoliv se Angular od AngularuJS výrazně liší, dá se pozorovat vzájemná podobnost, což se odráží v tom, že se tutoriály tematicky a ukázanou funkcionalitou velmi podobají. Tutoriál se znovu zaměřuje především na:

- Dvousměrné datové vazby a direktiva
- Komponenty
- DI
- Routování
- Získávání dat pomocí HTTP požadavků



Obr. 14 Angular aplikace Tour of Heroes – seznam hrdinů
Zdroj: Angular.io, 2017

Ani tentokrát není více prostoru věnováno formulářům, ale opět lze nalézt všechny informace o práci s formuláři, včetně příkladů, v dokumentaci (Angular.io, 2017). I přesto je tutoriál Angularu dostatečně obsáhlý a kladně lze hodnotit také použití TypeScriptu. Výsledný TS kód je, ve srovnání s JS, úspornější a lépe čitelný. Z provedené analýzy a testování Angularu vyplývá, že ho lze, stejně jako jeho předchůdce, použít pro refaktoring aplikace na SPA.

[9] TypeScript je open-source programovací jazyk vytvořený a spravovaný firmou Microsoft. Jedná se o nádstavbu nad jazykem JavaScript, která jej rozšiřuje o statické typování a další atributy, známé z objektově orientovaného programování (třídy, moduly, a další). (TypeScriptLang.org, 2016)

4.2.5 React

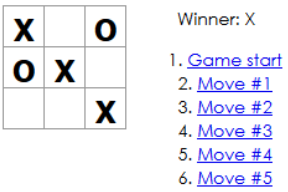
React (nebo také Facebook React) byl oficiálně vydán na konci května 2013, ale v praxi byl poprvé použit již v roce 2011 vývojáři Facebooku a následně v roce 2012 vývojáři Instagramu. Tyto 2 firmy se také dají považovat za hlavní vývojáře a přispěvovatele Reactu. (Wikipedia.org, 2016)

Některé vlastnosti Reactu (Facebook.GitHub.io, 2017):

- Základní knihovna se zaměřuje pouze na uživatelské rozhraní (vrstva View modelu MVC).
- Postaven na deklarativním programování.
- Modulární – při vývoji lze průběžně importovat potřebné komponenty.
- Vytváření znovupoužitelných komponent.
- Nezávislý na ostatních částech aplikace.
- Multiplatformní (využitím React Native).
- K dispozici syntax JSX pro přehlednější kód.

4.2.6 Testování Reactu

Tutoriál Reactu (verze 15.4.2), nacházející se na oficiálních stránkách (Facebook.GitHub.io, 2017), odpovídá tomu, že se Framework zaměřuje na uživatelské rozhraní a výsledná aplikace tak nemá žádný CRUD aspekt. Tutoriál uživatele provede tvorbou jednoduché hry piškvorek (viz obr. 15).



Obr. 15 React aplikace – piškvorky
Zdroj: Facebook.GitHub.io, 2017

Tento tutoriál je, ve srovnání s předchozími dvěma tutoriály, podstatně menší rozsahem. Jádrem tutoriálu jsou především komponenty, jejich vytváření a vzájemná interakce. Hlavní přednosti frameworku jsou představeny dostatečně, ovšem pro účely této práce tutoriál neukazuje, kromě zmíněného použití komponent, žádnou další z požadovaných funkcí. Po následně provedené analýze lze konstatovat, že při použití samotného Reactu:

- Nelze používat DI.
- Lze zpracovávat formuláře.
- Nelze routovat (možnost použít externí knihovnu).
- Nelze pracovat s HTTP požadavky (možnost použít externí knihovnu).

- Lze vytvářet komponenty.

Závěrem tedy je, že React není vhodným JS frameworkem pro refaktoring aplikace na SPA.

4.2.7 Vue.js

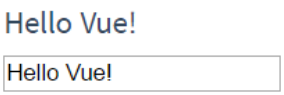
Vue.js je dílem vývojáře Yuxi You a pod názvem Vue.js byl framework poprvé vydán v prosinci 2013 (Vuejs.org, 2013). Vue.js[10] je framework pro tvorbu uživatelského rozhraní. V základu se tedy zaměřuje pouze na vrstvu View modelu MVC (Vuejs.org, 2017). Vue.js to kompenzuje tím, že je kompatibilní s ostatními knihovnami a snadno integrovatelný do již existujících projektů (Vuejs.org, 2017). Hlavní výhody Vue.js (Vuejs.org, 2017):

- Relativně snadno přístupný, lze ho začít rychle používat.
- Univerzální, kompatibilní a snadno integrovatelný.
- V základu velmi malý (23kb) a výkonný.

Vue.js je v mnoha ohledech velmi podobný Reactu, čemuž se věnuje na svých stránkách v rozsáhlém srovnání s ostatními frameworky (Vuejs.org, 2017)

4.2.8 Testování Vue.js

O testování frameworku Vue.js (verze 2.2.4) toho nelze napsat mnoho. Vue.js nemá žádný oficiální tutoriál a funkčnost je tedy představena pomocí průvodce (Vuejs.org, 2017), jehož kapitoly nejsou obsahově provázané a tím pádem se ani nevytváří ucelená aplikace (viz obr. 16). Za zmínku stojí fakt, že pro základní použití stačí importovat pouze JS knihovnu a není tak nutné využití lokálního serveru nebo Vue.js jakkoliv instalovat.



Obr. 16 Ukázka použití Vue.js – vazba popisku na textové pole
Zdroj: Vuejs.org, 2017

Při podrobnějším prozkoumání bylo zjištěno, že s Vue.js:

- Nelze používat DI.
- Lze zpracovávat formuláře.
- Lze routovat.
- Nelze pracovat s HTTP požadavky (možnost použít externí knihovnu).
- Lze vytvářet komponenty.

[10] Název frameworku je odvozen z anglické výslovnosti slova View. (Vue.js, 2017)

To odpovídá tomu, že Vue.js by mělo být jen jednou z několika použitých knihoven. Z toho důvodu a také např. kvůli absenci DI není Vue.js vhodným frameworkem pro plánovaný refaktoring aplikace na SPA.

4.2.9 Ember.js

Ember.js je komplexní a obsáhlý framework pro webové aplikace a zaměřuje se na všechny vrstvy modelu MVC (Emberjs.com, 2017). Poprvé byl framework vydán v prosinci 2011 (Emberjs.com, 2011). Ember.js spravuje samostatný tým lidí neboli Ember Core Team (Emberjs.com, 2017). Některé z předností Ember.js (Emberjs.com, 2017):

- Využití dynamických Handlebars[11] šablon.
- Vytváření nezávislých, znovupoužitelných komponent.
- Komplexní systém routování pro tvorbu vícestránkových aplikací.
- K dispozici je Ember CLI pro správu struktury projektu a snadnou integraci rozšíření. (Ember-CLI.com, 2017)

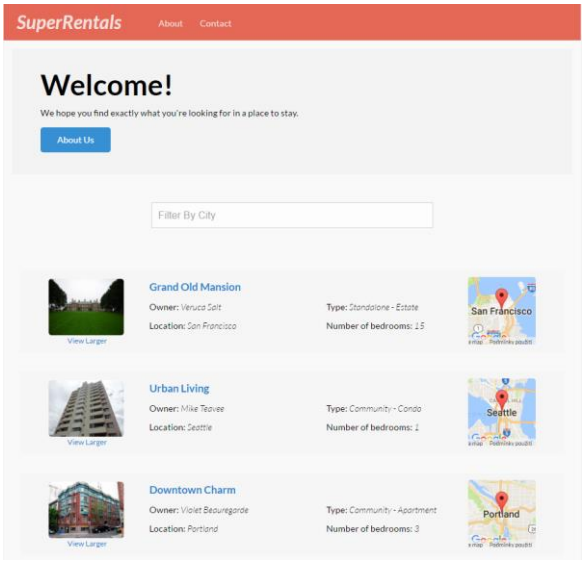
4.2.10 Testování Ember.js

V rámci oficiálního tutoriálu Ember.js (Emberjs.com, 2017), se vytváří relativně komplexní SPA, pracující s každou vrstvou architektury MVC, což odpovídá účelu frameworku. Základ aplikace znovu tvoří UI vzor master-detail a výsledná aplikace obsahuje seznam nemovitostí k pronájmu (viz obr. 17) u nichž lze zobrazit detail s podrobnějšími informacemi. Tutoriál je velmi obsáhlý (což se projevuje např. na komplexní struktuře projektu, která ze všech testovaných frameworků využívá nejvíce souborů) a postupně se věnuje mj.:

- Routování.
- Vytváření komponent.
- Použití Handlebars šablon.
- Použití HTTP požadavků.
- Použití DI.
- Integraci Google Maps API.
- Nasazení aplikace do produkce.

Ani tutoriál pro Ember.js se podstatněji nevěnuje práci s formuláři, ale i tentokrát podrobnější prozkoumání dokumentace odhalilo, že se Ember.js dokáže navázat na formulářové prvky (Emberjs.com, 2017). Ember.js tedy také vyhovuje všem nastaveným kritériím a mohl by být použit k refaktorování aplikace na SPA.

[11] Handlebars.js je JS knihovna pro tvorbu sémantických šablon. Handlebars jsou založeny na Mustache šablonách, jež rozšiřují a jsou s nimi vzájemně kompatibilní. (Handlebars.com, 2017)



Obr. 17 Ember.js aplikace SuperRentals – úvodní stránka, přehled nemovitostí
Zdroj: Emberjs.com, 2017

4.2.11 Volba frameworku pro vytvoření single-page aplikace

Na začátku podkapitoly jsem definoval požadavky, kterým musí potenciálně použitý JS framework vyhovět. Pomocí těchto požadavků jsem následně analyzoval a otestoval pět často používaných JS frameworků. Výsledkem testování je, že k refaktorování aplikace na SPA, je možné použít 3 JS frameworky: AngularJS, Angular a Ember.js.

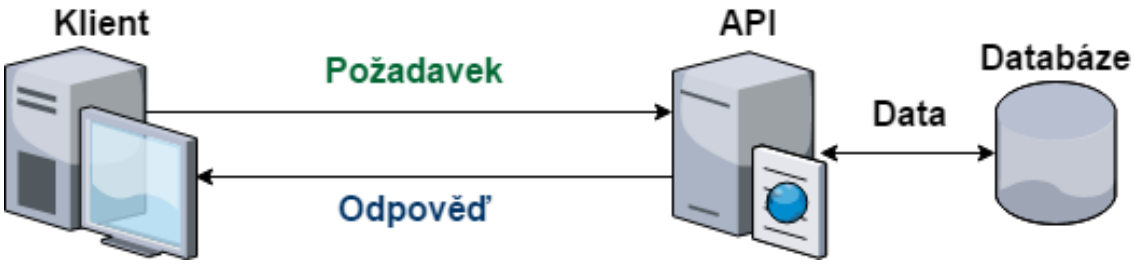
Ačkoliv lze použít AngularJS, který používá rozsáhlá komunita vývojářů, primárním JS frameworkem Googlu je v současnosti již Angular a dá se očekávat, že má ve vývoji přednost. Při testování se lépe pracovalo s Angularem a také z toho důvodu se zdá být celkově lepší volbou.

Zbývá tedy zvolit mezi Angularem a Ember.js. Tyto JS frameworky se od sebe navzájem výrazně liší. Pro Angular hovoří fakt, že se o jeho vývoj stará tak velká společnost, jakou je Google. Také za relativně krátkou dobu své existence má již na GitHubu lepší hodnocení než Ember.js (GitHub.com, 2017). Výhodou Ember.js může být mj. to, že se používá už od roku 2011 a má pravděpodobně zkušenější uživatelskou základnu. Při testování byla intuitivnější práce s Angularem a i proto se opět jeví jako lepší volba. Rozhodl jsem se, že pro refaktoring aplikace na SPA použiji JavaScriptový framework Angular.

4.3 REST API

V předchozí podkapitole jsem vybral Angular jako JS framework, který použiji pro refaktorování aplikace. To s sebou ale nese určité nároky na další části aplikace. Momentálně nejpodstatnější z těchto nároků souvisí s tím, že nelze data přenášet přímo mezi aplikací a databázovým systémem. Angular ovšem může se svým okolím komunikovat prostřednictvím HTTP požadavků.

Pro přenos dat mezi aplikací a databází proto vytvořím REST API, která bude s aplikací používající Angular komunikovat pomocí HTTP požadavků a potřebná data zprostředkovávat. Nová refaktorovaná aplikace bude tedy rozdělena na klientskou část používající Angular a serverovou část poskytující API (viz obr. 18).



Obr. 18 Komunikace mezi klientem, API a databází v nové aplikaci

4.3.1 Representational State Transfer

Representational State Transfer (REST) je architektonický styl pro distribuované systémy (Fielding, 2000, s. 76), jenž identifikoval problémy v rámci HTTP/1.0 a byl základem pro specifikování HTTP/1.1 (Fielding, 2000, s. 116). REST definuje 6 hlavních principů (zásad) pro vytváření webového rozhraní (Fielding, 2000, s. 78-85):

1. Klient-server (Client-Server)

   Oddělení odpovědnosti. Neboli záležitosti uživatelského rozhraní by měly být odděleny od záležitostí datového úložiště.

2. Bezstavovost (Stateless)

   Každý požadavek mezi klientem a serverem musí obsahovat všechny informace potřebné k jeho vykonání.

3. Mezipaměť (Cache)

   Každá odpověď požadavku, která obsahuje data, musí obsahovat také informaci o tom, jestli se data mají nebo nemají uložit do mezipaměti.

4. Jednotné rozhraní (Uniform Interface)

   Klient a server spolu komunikují standardizovanou formou nezávislou na rozdílech jednotlivých implementací.

5. Vrstvený systém (Layered System)

   Systém je rozdělen do spolupracujících vrstev, jež jsou od sebe navzájem funkčně odděleny.

6. Kód na vyžádání (Code-On-Demand)

   Možnost rozšíření funkcionality klienta stáhnutím a spuštěním části kódu. Tato zásada je jako jediná nepovinná.

4.3.2 Návrh nové REST API

Postup návrhu a vytváření REST API byl částečně inspirován radami obsaženými v knize Undisturbed REST (Stowe, 2015), která se touto problematikou zabývá. Podrobné informace o navrhované API se nachází také v elektronické dokumentací, která je k dispozici na webové adrese: http://docs.curriculumgeneratorapi.apiary.io/

4.3.2.1 Zdroje (Resources)

Ačkoliv se bude aplikace výrazně měnit, většina zpracovávaných dat zůstává kategoricky stejná. Stále bude především potřeba ukládat a načítat informace o uživateli a životopis společně s jeho částmi přidanými v jednotlivých krocích tvorby životopisu.

Jednotlivé zdroje je proto potřeba jednoznačně identifikovat v rámci API. Na základě těchto požadavků jsou definovány zdroje v tab. 1, pro něž bude API poskytovat přístupové body, na které následně může klientská aplikace posílat HTTP požadavky.

Tab. 1 Datové zdroje REST API

Jméno	API ID	Popis
Uživatelé	users	Uživatelské účty
Životopisy	curricula	Základní informace - 1. krok
Vzdělání	educations	2. krok
Zaměstnání	employments	3. krok
Kurzy	courses	4. krok
Jazyky	languages	5. krok
Členství	memberships	6. krok
Výzkum	researches	7. krok
Publikace	publications	8. krok
Výuka	tutorships	9. krok
Technické schopnosti	skills	10. krok
Dodatečné informace	additionals	11. krok

Více informací o jednotlivých zdrojích se nachází také v dokumentaci API.

4.3.2.2 Specifikace

S ohledem na 4. princip REST architektury je potřeba definovat standard – jednotnou formu, kterou spolu budou klient a server komunikovat. Jako prostředek komunikace již z okolností vyplynul protokol HTTP, který odpovídá potřebám aplikace. Co se týče typu obsahu posílaných požadavků a odpovědí, doporučuje Stowe (2015, s. 76) JSON, s kterým navíc také Angular umí ve výchozím stavu pracovat.

Dalším krokem je návrh struktury posílaného obsahu. Aby klientská aplikace a server mohly obdržená data dále zpracovat, potřebují se spolehnout na to, že jsou

data konkrétně uspořádána. Nabízí se řešení návrhem vlastní struktury obsahu, která by přesně odpovídala potřebám aplikace. Ovšem jako pravděpodobně lepší alternativa se nabízí použití některé, z již existujících specifikací. Výhody tohoto řešení se projeví např. pokud se:

- API bude v budoucnu rozšiřovat.
- API stane veřejným a budou ho využívat i externí aplikace.
- O aplikaci a API bude starat více vývojářů.

Existuje mnoho různých specifikací JSON dokumentu. Mezi nejvíce používané, které zmiňuje také Stowe (2015, s. 124), patří:

- JSON API[12]
- HAL[13]
- JSON-LD[14]

Jako nejlepší ze jmenovaných se jeví specifikace JSON API, která je nejlépe hodnocená na GitHubu (JSONAPI.org, 2015) a mj. definuje také formát chybových odpovědí (JSONAPI.org, 2015). API tedy bude využívat tuto specifikaci. Příklad požadavku, který žádá o vrácení jednoho vzdělání s ID 1:

```
GET /educations/1
```

Odpověď zformátovaná podle JSON API obsahující požadovaný objekt vzdělání:

```
{
   "data": {
      "type": "educations",
      "id": "1",
      "attributes": {
         "obdobiOd": 09/2012,
         "obdobiDo": "06/2015",
         "kvalifikace": "Bakalářské",
         "obor": "Ekonomická informatika",
         "organizace": "Mendelova univerzita v Brně"
      }
      "relationships": {
         "curriculum": {
            "data": {"id": "1", "type": "curricula"}
         }
      }
   }
}
```

[12] http://jsonapi.org/

[13] http://stateless.co/hal_specification.html

[14] http://json-ld.org/

4.3.2.3 Autentizace a autorizace

Všechny data, s kterými aplikace pracuje jsou citlivé údaje. Ať už se jedná o uživatelské údaje nebo údaje související s životopisem, nesmí se k údajům dostat nepovolaná osoba (k čemuž se také aplikace zavazuje v podmínkách, které uživatel schvaluje). Vzhledem k tomu, že z podstaty věci API nemá kontrolu nad tím, který z klientů posílá požadavek žádající specifická data, nelze otevřeně zpřístupnit přístupové body ke zdrojům API.

Nejprve je tedy nutné provést autentizaci, kterou se klientská aplikace jednoznačně identifikuje – v tomto případě přihlášením uživatele. Tím se zaručí, že požadavky posílá klient, kterému byl přidělen přístup ke zdrojům API. Klient ale může žádat pouze o zdroje vytvořené právě přihlášeným uživatelem – tím je zajištěna autorizace.

Druhý princip REST architektury udává, že každý požadavek musí obsahovat všechny informace potřebné pro jeho vykonání. Z toho vyplývá, že není možné, aby se klientská aplikace autentizovala pouze při přihlášení uživatele. Každý požadavek[15] v sobě musí obsahovat také identifikaci uživatele.

V současnosti běžným řešením tohoto problému je, jak zmiňuje Stowe (2015, s. 59), použití tokenů, které v sobě obsahují všechny potřebné informace nejen o uživateli, ale třeba také délku validity konkrétního tokenu apod. Stowe (2015, s. 62) zmiňuje jako vhodný způsob zpracování tokenů pomocí specifikace OAuth2[16]. Pro veřejné API je tato velmi komplexní specifikace vhodnou volbou, ovšem pro účely této aplikace v současnosti postačí použití specifikace JSON Web Token[17] (JWT).

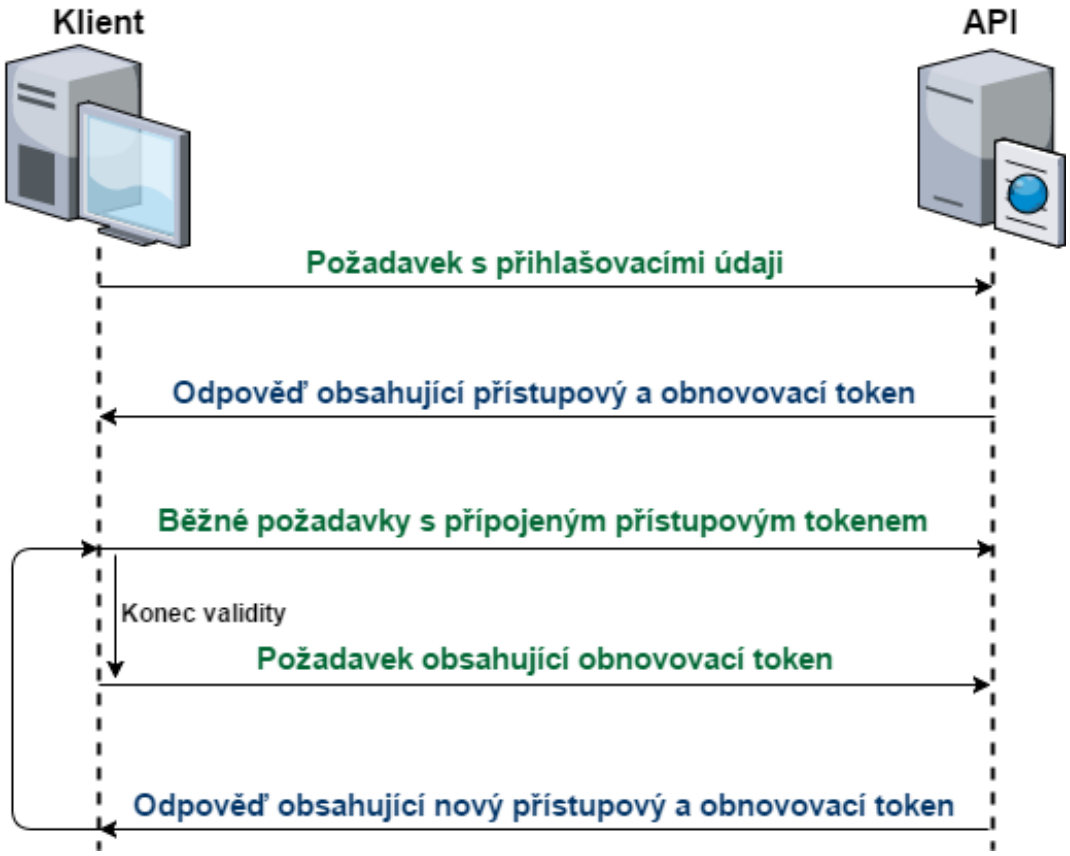
Systém použití tokenů do API začlením vytvořením nových (nezdrojových) přístupových bodů, které budou sloužit k autentizaci uživatele. Proces, jak systém autentizace a autorizace pomocí tokenů funguje (viz obr. 19), je následovný:

1. Při přihlášení uživatele klient zašle požadavek s přihlašovacími údaji na přístupový bod *Přihlášení* (login).
2. API v reakci na úspěšné přihlášení vygeneruje přístupový („access“) token (s kratší dobou validity) a obnovovací („refresh“) token (s delší dobou validity, ale na jedno použití), které zašle v odpovědi.
3. Klient ke každému požadavku připojí přístupový token, jehož validitu API zkontroluje a pomocí nějž je uživatel identifikován.
4. Když vyprší validita přístupového tokenu, klient automaticky zašle požadavek obsahující obnovovací token na přístupový bod *Obnovení* (refresh).
5. API vygeneruje nové tokeny (přístupový i obnovovací), které pošle v odpovědi.

[15] Výjimky jsou: požadavky metody OPTIONS, vytvoření a přihlášení uživatele.

[16] https://oauth.net/2/

[17] https://jwt.io/



Obr. 19 Systém autentizace a autorizace pomocí JWT.

Díky tomuto systému se uživatel nemusí přihlašovat vždy když skončí validita přístupového tokenu. Přitom je zajištěna určitá úroveň bezpečnosti. Potenciální útočník, který odchytne náhodný požadavek, má k dispozici pouze přístupový token s relativně krátkou dobou validity. Další informace se nachází také v dokumentaci API a v podkapitole 5.3.

4.4 Změny databáze

Jak bylo uvedeno v podkapitole 3.2.1, současná databáze by měla dostát jistých změn a z provedeného návrhu REST API vyplývá, že je potřeba vytvořit jednu novou tabulku (viz podkapitola 4.4.1).

4.4.1 Tabulky

Při návrhu REST API bylo definováno 12 datových zdrojů. Každý z těchto zdrojů bude v databázi zastoupen jednou tabulkou. Přibyde tedy jedna nová tabulka pro dodatečné informace vyplňované v předposledním kroku tvorby životopisu. Dodatečné informace se v současnosti ukládají do tabulky *Zivotopis*, ale vzhledem k tomu, že se s daty pracuje v rozdílných částech tvorby životopisu, jeví se jako vhodné tyto informace oddělit.

Poslední, třináctou a zároveň nezdrojovou tabulkou databáze bude tabulka pro jazyk životopisu. Z toho vyplývá, že značný počet tabulek (jež refaktorovaná aplikace nebude potřebovat) ubyde. Důvody odstranění tabulek:

- Feedback, FeedbackTyp, Fotografie, ZivotopisFotografie, Preklad a PraceTyp

  Neobsahují žádná data, nepoužívají se.

- SysJazyk a Stranka

  Lokalizace bude řešena v klientské aplikaci.

- Uroven, Kvalifikace a ZivotopisStav

  Nemají žádný podstatný účel a mohly být transformovaný na sloupec.

- ZivotopisTyp, Role

  Obsahují jen jeden záznam. Je jen jeden typ životopisu a jedna role.

- Log

  Ukládaná data se k ničemu nevyužívají a tabulka tak nemá podstatný význam.

- PlatbaSMS

  Platba pomocí SMS nebude nadále používána. V budoucnu bude případně vytvořena tabulka pro nový způsob platby.

4.4.2 Názvy tabulek a sloupců

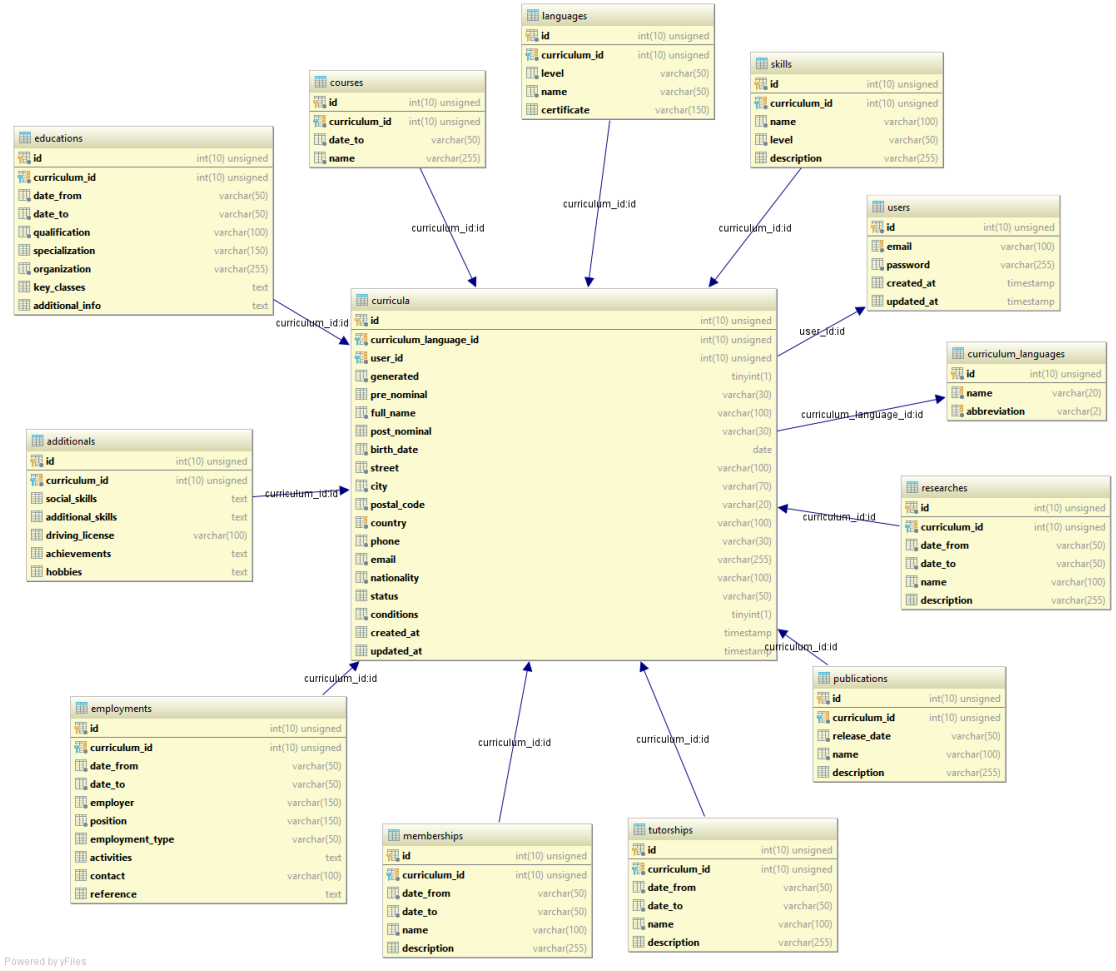
V současné databázi jsou tabulky a sloupce pojmenovány česky. S přihlédnutím ke konvencím je pro budoucí využití lepší použít anglické názvy. Nové názvy tabulek proto odpovídají nově definovaným zdrojům API (viz tab. 2).

Tab. 2 Změna názvů tabulek v databázi

Starý název	Nový název
uzivatel	users
zivotopis	curricula
vzdelani	educations
prace	employments
kurz	courses
jazyk	languages
clenstvi	memberships
vyzkum	researches
publikace	publications
vyuka	tutorships
techschop	skills
zivotopisjazyk	curriculum_languages

4.4.3 Schéma nové databáze

Schéma databáze po provedení všech uvedených změn ukazuje obr. 20.



Obr. 20 Schéma databázového modelu po provedení změn.

5 Implementace API

V předcházející kapitole byly navrhnuty rozsáhle změny, které se týkají všech částí aplikace. Podstatně se změnila architektura aplikace, kdy původní ucelená aplikace, postavená na architektuře MVC, byla rozdělena na dvě části – klientskou aplikaci a serverovou část poskytující API. Tato kapitola se věnuje implementaci navrhnutého API rozhraní.

Části současné aplikace, které zahrnují uživatelské rozhraní a jeho funkční logiku, budou implementovány jako součást klientské aplikace. V druhé části aplikace, pracující především s daty, se tím pádem vytrácí význam použití full-stack frameworku. Pro Nette sice jsou k dispozici neoficiální knihovny pomáhající s implementací REST API[18], ale vzhledem k tomu, že klientská část aplikace bude přepracována pomocí Angularu, většina funkcí a předností Nette by nebyla využita. Bylo by však v rámci možností vhodné jako základ zachovat PHP a díky tomu znovu použít funkce (s případnými úpravami), které budou i nadále potřeba (např. generování PDF).

Nabízí se řešení pomocí tzv. „micro-frameworku“[19]. Existují totiž micro-frameworky, pro které je vytváření REST API jejich primárním účelem. Příkladem takových frameworků jsou např. Slim[20], Silex[21] a Lumen. Právě poslední jmenovaný, Lumen, se jeví jako vhodná volba. Lumen je odnoží full-stack frameworku Laravel a lze díky tomu v Lumenu použít např. databázovou vrstvu, integrovat posílání e-mailů nebo využít některé knihovny, jež byly primárně vytvořeny pro Laravel (Lumen.Laravel.com, 2017). To je pro vývoj navrhované API výhodou a Lumen jsem tedy pro implementaci API použil.

5.1 Přístupové body

Potřebné přístupové body (uvedené v podkapitole 4.3.2.1) lze v Lumenu implementovat definováním *rout*. Na tyto *routy* lze následně posílat požadavky a v závislosti na metodě[22] HTTP požadavku se volají funkce přiřazeného *Controlleru*, který požadavek dále zpracovává.

```
1  $app->group(['prefix' => 'curricula'], function () use ($app) {
2      $app->options('/', 'CurriculumController@optionsCollection');
3      $app->options('/{id}', 'CurriculumController@optionsResource');
4
5      $app->group(['middleware' => 'auth', 'as' => 'curricula'],
6  function () use ($app) {
```

[18] Např. https://github.com/drahak/Restful

[19] Micro-frameworky nebo také „nonfull-stack“ jsou frameworky, které se zaměřují pouze na některé činnosti ve vývoji. (Blog.AppDynamics.com, 2015)

[20] https://www.slimframework.com/

[21] https://silex.sensiolabs.org/

[22] Použité metody – OPTIONS, GET, POST, DELETE nebo PATCH

```
7         $app->get('/', 'CurriculumController@getCollection');
8         $app->post('/', 'CurriculumController@createResource');
9
10        $app->get('/{id}', 'CurriculumController@getResource');
11        $app->patch('/{id}', 'CurriculumController@updateResource');
12        $app->delete('/{id}', 'CurriculumController@deleteResource');
13    });
14 });
```

Řádek 1: Definice skupiny *rout* s prefixem *curricula*[23] neboli přístupový bod pro zdroj *Životopis*.

ř. 2 a 3: Definice akcí pro požadavek metody OPTIONS[24]. Pro kolekci zdrojů (**2**) a pro konkrétní zdroj specifikovaný pomocí ID (**3**).

ř. 5 a 6: Zabezpečení *rout*, skupina akcí na následujících řádcích je dostupná pouze pro autentizované požadavky.

ř. 7 a 8: Definice akcí pro požadavky metody GET a POST pro kolekci zdrojů. Odpověď na úspěšný požadavek GET obsahuje kolekci zdrojů (**7**). Úspěšný požadavek POST vytváří nový zdroj typu *Životopis* (**8**).

ř. 10-12: Definice akcí pro požadavky metod GET, PATCH a DELETE pro konkrétní zdroj specifikovaný pomocí ID. Odpověď na úspěšný požadavek GET obsahuje specifikovaný zdroj (**10**). Požadavek PATCH mění atributy specifikovaného, již existujícího zdroje (**11**). Požadavek DELETE maže specifikovaný zdroj (**12**).

Odpovídajícím způsobem jsou definovány také všechny další přístupové body API.

5.2 Middleware

Požadavek, který API obdrží na některém z přístupových bodů, není hned zpracován *Controllerem*. Korektnost požadavku je nejprve ověřena pomocí *middleware*. API využívá celkem 3 *middleware* kontrolující požadavek:

1. `<RequestHeaders>` kontroluje správnost hlaviček („headers“) požadavku. Především ve spojitosti s pravidly specifikace JSON API.
2. `<RequestParameters>` kontroluje parametry požadavku. Pokud požadavek obsahuje nepovolené parametry, není dále zpracován.
3. `<Authenticate>` je součástí procesu autentizace. U zabezpečených *rout* kontroluje, jestli je požadavek autorizován (viz podkapitola 5.3).

[23] Pokud by se API nacházela např. na URL adrese https://www.curriculum-generator.com/api, tak přístupový bod *Životopis* by byl dostupný na URL adrese https://www.curriculum-generator.com/api/curricula

[24] Angular automaticky zasílá požadavky metody OPTIONS před požadavky ostatních metod.

Poslední *middleware* API se využívá při odesílání odpovědi, kdy *middleware* `<ResponseHeaders>` nastaví všechny potřebné hlavičky odpovědi.

5.3 Autentizace a autorizace

5.3.1 Přihlášení uživatele

Prvním krokem autentizace je přihlášení uživatele. API zkontroluje údaje zaslané v požadavku a v případě správnosti údajů vygeneruje, pomocí knihovny Lcobucci JWT[25], přístupový a obnovovací token. Přístupový token generuje následující funkce:

```
15  public static function generateToken(int $user_id): Token {
16     $tokenBuilder = new Builder();
17     $signer       = new Sha256();
18
19     $token = $tokenBuilder->setIssuer(route('api'))
20                       ->setIssuedAt(time())
21                       ->setNotBefore(time())
22                       ->setExpiration(time() + self::ID_TOKEN_TTL)
23                       ->set('uid', $user_id)
24                       ->sign($signer, env('JWT_SECRET'))
25                       ->getToken();
26
27     return $token;
28  }
```

Řádek 15: Parametrem je ID uživatele.

ř. 16 a 17: Vytvoření objektů knihovny Lcobucci JWT pro sestavení tokenu (**16**) a šifrování (**17**).

ř. 19-25: Vytváření tokenu. Nastavení: poskytovatele tokenu (tato API) (**19**), času přidělení tokenu (**20**), času od kdy je token platný (**21**), čas expirace validity tokenu neboli životnost (**22**), uživatelského ID (**23**), zašifrovaného tajného podpisu (pro kontrolu validity) (**24**).

Obdobným procesem se vytváří také obnovovací token. Navíc se ale zajišťuje, že lze obnovovací token použít jen jednou.

```
29  $id = bin2hex(random_bytes(32));
30  Cache::put(
31     self::TOKEN_CACHE . $user_id,
32     Hash::make($id),
```

[25] Lcobucci JWT je knihovna pro práci s JSON Web Tokeny a JSON Web Signaturami. (Cobucci, 2014)

```
33      Carbon::now()->addSeconds(self::REFRESH_TOKEN_TTL)
34  );
```

Řádek 29: Vytvoření náhodného, unikátního[26] ID. Toto ID je připojeno k tokenu v rámci procesu vytváření.

ř. 30-34: ID tokenu se pro přihlášeného uživatele uloží do mezipaměti na tak dlouho, jak dlouhou má obnovovací token životnost.

Oba tokeny jsou následně poslány v odpovědi, kterou dále zpracuje klientská aplikace (viz podkapitola 6.3.1).

5.3.2 Autentizace požadavku prostřednictvím tokenu

Jakmile API obdrží požadavek na některou ze zabezpečených *rout*, začíná proces autentizace. Autentizaci provádí poskytovatel `<AuthServiceProvider>` (ASP):

4. ASP zkontroluje, že požadavek obsahuje autorizační hlavičku, a že má obsah hlavičky správný formát.
5. ASP pomocí knihovny Lcobucci JWT ověří validitu tokenu pomocí parametrů nastavených při generování tokenů (jestli životnost tokenu nevypršela, jestli je správně podepsaný apod.).
6. ASP automaticky přihlásí uživatele jehož ID token obsahoval. Tím je požadavek autorizován.
7. Požadavek může být, v závislosti na jeho metodě, autorizován ještě také v příslušné funkci *controlleru*. Tam se ověří, zdali je přihlášený uživatel majitelem manipulovaného zdroje.

5.3.3 Obnova tokenu

Při požadavku na obnovení tokenu se ověřuje pouze validita obnovovacího tokenu. Součástí tohoto procesu je také ověření existence uloženého ID tokenu v mezipaměti. V případě shody se ID vymaže (token už nelze znovu použít), načež se vygenerují[27] a odešlou nové tokeny. V opačném případě proces končí a uživatel se musí znovu přihlásit.

5.3.4 Odhlášení uživatele

Požadavek na odhlášení se autentizuje klasickým způsobem, načež se z mezipaměti odstraní ID obnovovacího tokenu pro autentizovaného uživatele. Tím pádem už nelze použít příslušný obnovovací token pro vygenerování nového přístupového tokenu.

---

[26] http://php.net/manual/en/function.random-bytes.php

[27] Proces generování nových tokenů je identický s procesem generování tokenů při přihlášení uživatele.

5.4 Zpracování požadavku

Pokud se požadavek dostal až do *controlleru*, nemá závadný obsah[28] a manipuluje pouze s daty, ke kterým má uživatel přístup, je požadavek zpracován a výsledek odeslán odpovědí ve formátu podléhajícímu JSON API specifikaci. S formátováním obsahu odpovědí pomáhá knihovna Neomerx JSON API[29]. Základem použití je příprava schémat, které se předají funkci knihovny. Pomocí nich pak dokáže knihovna zpracovávat předaná data a vytvářet správně formátovaný obsah odpovědi.

```
35  public function getAttributes($education): array {
36
37      $attributes = [
38          'dateFrom'       => $education->date_from,
39          'dateTo'         => $education->date_to,
40          'qualification'  => $education->qualification,
41          'specialization' => $education->specialization,
42          'organization'   => $education->organization,
43          'keyClasses'     => $education->key_classes,
44          'additionalInfo' => $education->additional_info
45      ];
46
47      return $attributes;
48  }
```

Řádek 35-48: Schéma vzdělání – načítání atributů odpovědi.

```
49  public function getRelationships($education, $isPrimary, array
50  $includeRelationships) {
51      return [
52          'curriculum' => [
53              self::DATA         => $education->curriculum,
54              self::SHOW_RELATED => false,
55              self::SHOW_SELF    => false
56          ],
57      ];
58  }
```

Řádek 49-58: Schéma vzdělání – načítání vztahů zdroje pro odpověď.

```
59  public static function encodeResource($data): string {
60      $encoderOptions = new EncoderOptions(0, route('api'));
61      $encoder        = Encoder::instance(
62          [Education::class => EducationSchema::class],
63          $encoderOptions
```

[28] V případě POST a PATCH požadavků se validuje také obsah požadavku.

[29] Neomerx JSON API je „framework agnostic“ knihovna pro vytváření odpovědí zformátovaných podle JSON API specifikace. (Neomerx, 2017)

```
64      );
65
66      $json = $encoder->encodeData($data);
67
68      return $json;
69  }
```

Řádek 60: Vytvoření nastavení enkodéru.

ř. 61-64: Vytvoření instance enkodéru, při čemž se předá pole použitých schémat (**62**) a nastavení (**63**).

ř. 66: Zakódování neboli formátování dat.

ř. 68: Funkce vrací zformátovaný JSON, který se pošle v odpovědi.

Výsledkem tohoto procesu by byl JSON, jenž se nachází v podkapitole 4.3.2.2.

5.5 Generování PDF

Proces generování PDF šlo částečně převzít z původní aplikace, kde byla použita knihovna mPDF. V Lumenu lze využít knihovnu Laravel PDF[30], která použití mPDF ještě více usnadňuje. Bylo už jen potřeba obsah původních šablon pro generování PDF přizpůsobit Blade[31] šablonám Laravelu, které lze používat také v Lumenu.

API má pro generování speciální přístupový bod, na který se zašle požadavek na vygenerování PDF, následovaný běžným procesem autentizace. Dále již *v controlleru* začíná proces generování PDF, kdy se z databáze načte životopis a ověří se autorizace uživatele. Následně se již všechny data související s životopisem předají šabloně a začíná samotný proces generování.

```
70  $pdf = mPdf::loadView('pdf.curriculum-pdf', $data, [],
71  self::config($curriculum));
72
73  $resource       = new Pdf();
74  $resource->file = base64_encode($pdf->output());
75
76  $curriculum->generated = true;
77  $curriculum->save();
78
79  $json = Encoder::encodeResource($resource);
80
81  return response($json);
```

(část funkce vynechána)

[30] Laravel PDF je knihovna pro snadné generování PDF v Laravelu. (Ravensborg-Gjertsen, 2016)

[31] https://laravel.com/docs/5.4/blade

Řádek 70 a 71: Funkce knihovny Laravel PDF – načtení šablony, dat a nastavení pro generování PDF.

ř. 73 a 74: Generování životopisu (vytvoření objektu knihovny Laravel PDF) (**73**) a načtení životopisu zakódovaného jako base64 řetězec do proměnné (**74**).

ř. 76 a 77: Uložení informace o generování životopisu do databáze

ř. 79: *JSON API* formátování obsahu odpovědi.

ř. 81: Zaslání odpovědi klientské aplikaci, která s obsahem dále pracuje (viz podkapitola 6.44).

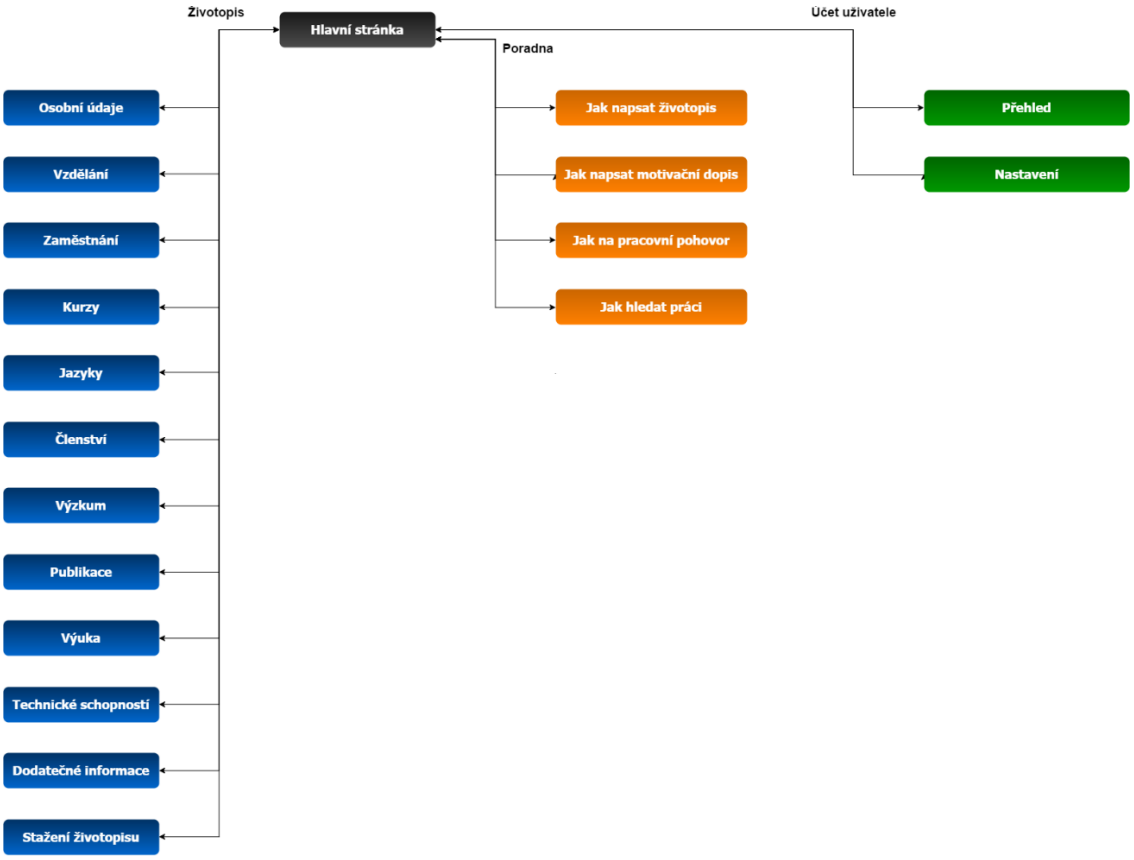
6 Implementace klientské aplikace

Pro implementaci klientské aplikaci byl zvolen v podkapitole 4.2 JS framework Angular. Vzhledem k rozsáhlým změnám ve všech oblastech aplikace je potřeba začít znovu vytvářet uživatelské rozhraní, včetně veškeré funkčnosti a aplikační logiky.

6.1 Architektura aplikace

Architektura aplikace (viz obr. 21) se, kromě zřetelného rozšíření o administraci uživatelského účtu, podstatně změnila i v dalších ohledech. Např. pokud je uživatel již přihlášený a nachází se na hlavní stránce, platí následující:

- V původní aplikaci musel uživatel pro vygenerování životopisu (potřeba nejvíce akcí) udělat v ideálním případě 13 akcí. Tzn. také 13 načtení aplikace.
- V refaktorované aplikaci se uživatel dostane na jakékoliv místo aplikace nejvýše 4 akcemi (a aplikace se nemusí nikdy znovu načítat).



Obr. 21 Webová architektura refaktorované aplikace Curriculum Generator.

6.2 Uživatelské rozhraní

V *Angularu* lze pro konstrukci uživatelského rozhraní využívat klasické HTML šablony a kaskádové styly (CSS). Ovšem vytvářet uživatelské rozhraní úplně od základu by nebylo efektivní. Vhodnější je využít některou z knihoven pro tvorbu uživatelského rozhraní, která práci výrazně usnadní.

Jednou takovou knihovnou, jedinou v aplikaci použitou, je Angular Material. Komponenty, které knihovna poskytuje jsou vytvořeny na základě principů materiálového designu od Googlu[32]. Knihovna je navíc vyvinuta speciálně pro použití v Angularu (Material.Angular.io, 2016). Angular Material umožňuje v HTML šablonách použití speciálních tagů, pomocí nichž lze přidávat jednotlivé komponenty knihovny (Material.Angular.io, 2017).

Uživatelské rozhraní bylo tedy vytvořeno především pomocí knihovny Angular Material a následně již byly udělány vlastní úpravy prostřednictvím CSS. Optimalizována byla také responzivita a refaktorovaná aplikace se tak dokáže v rámci možností přizpůsobit různým rozlišením obrazovky. Dolní hranice optimalizace je rozlišení o šířce 400px.

6.2.1 Hlavní stránka

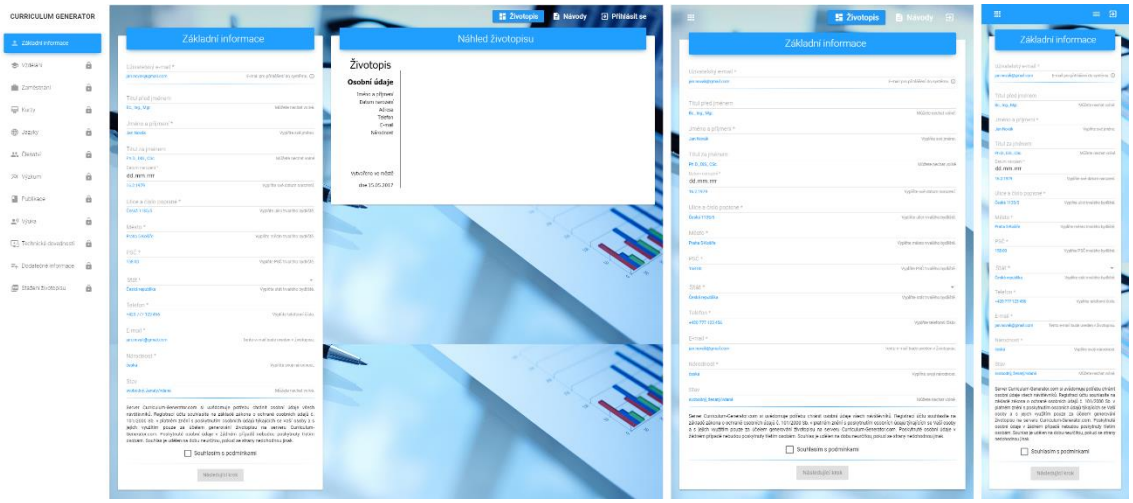
Hlavní stránka dostála podstatných změn, odpovídajících návrhu v podkapitole 4.1 (viz obr. 22). Hlavním účelem nové hlavní stránky je především zaujmout a předat podstatné informace.

[32] https://material.io/guidelines/material-design/introduction.html

Obr. 22 Hlavní stránka refaktorované aplikace v rozlišení o šířce 1920px, 768px a 400px respektive.

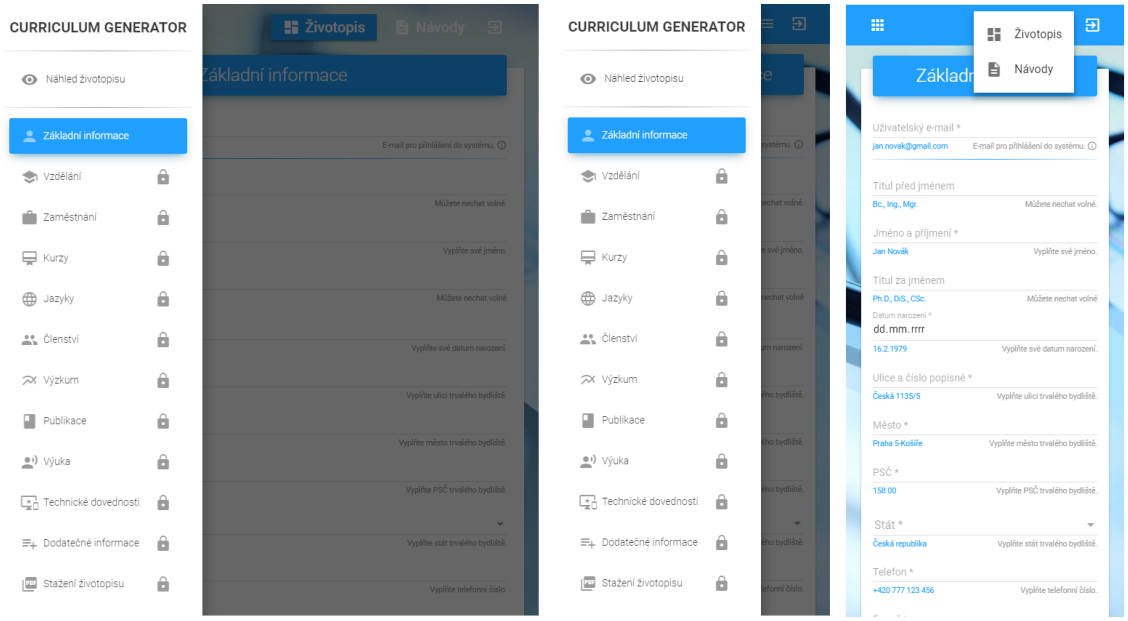
6.2.2 Životopis

Přepracovanou část aplikace pro tvorbu životopisu lze vidět na obr. 23 (duplikace pozadí pouze na obr.) a obr. 24. Při menším rozlišení se boční menu a náhled životopisu automaticky skryjí. Boční menu se otevírá pomocí tlačítka v levém horním rohu uživatelského rozhraní. Náhled lze na menších rozlišeních zobrazit pomocí tlačítka, které se nachází v bočním menu.



Obr. 23 Refaktorovaná stránka pro vytváření životopisu v rozlišení o šířce 1920px, 768px a 400px respektive.

Formátování v náhledu životopisu odpovídá formátování ve výsledném PDF. V rámci možností tak věrohodně reprezentuje výsledný životopis.

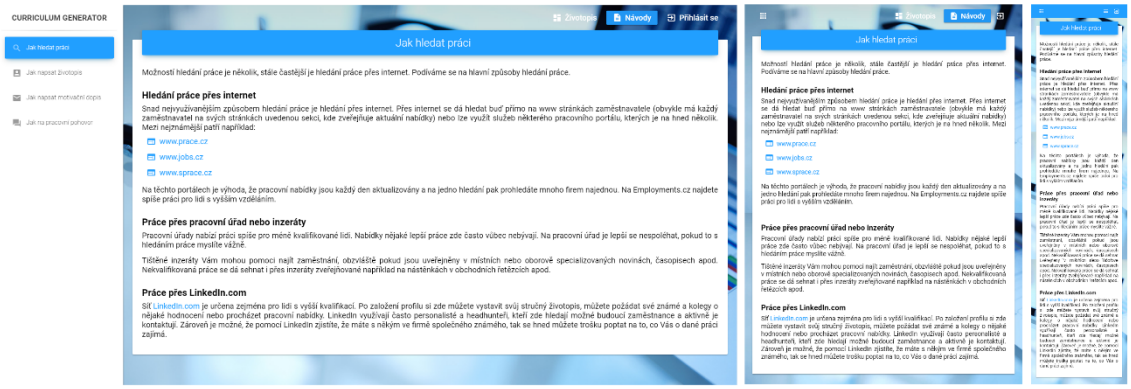


Obr. 24 Otevřené boční menu při rozlišení o šířce 768px a 400px respektive, poslední část obrázku ukazuje otevřené hlavní menu při rozlišení o šířce 400px.

Boční menu je vytvořeno pomocí komponenty knihovny Angular Material, která práci s bočním menu výrazně usnadňuje. Tlačítka jsou již vytvořena vlastními úpravami.

6.2.3 Návody

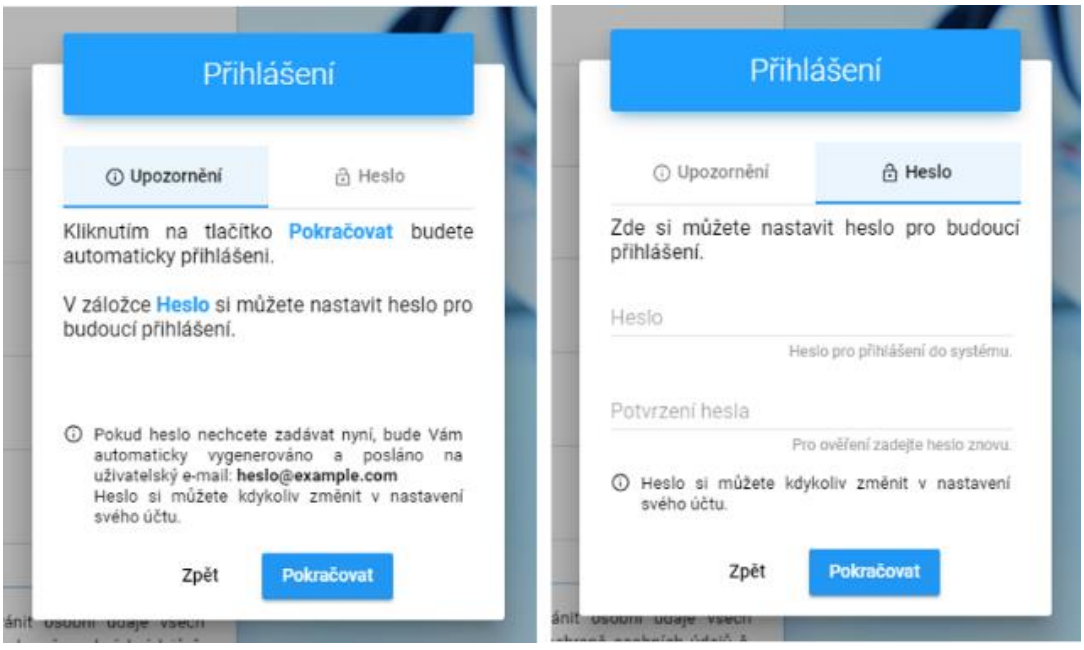
Refaktorovanou stránku s návody lze vidět na obr. 25. I zde je k dispozici boční menu, pomocí kterého se lze přepínat mezi jednotlivými návody.



Obr. 25 Refaktorovaná stránka s návody v rozlišení o šířce 1920px, 768px a 400px respektive.

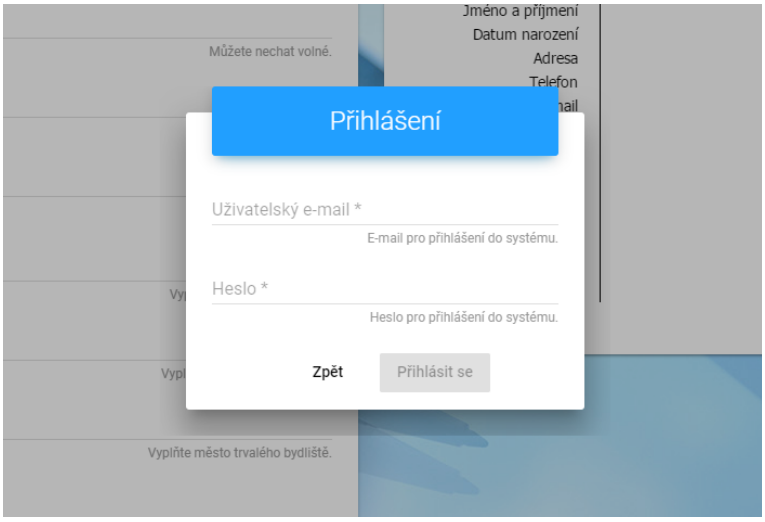
6.2.4 Uživatel

Nový uživatel aplikace si může vytvořit účet obdobným způsobem, jako tomu bylo v původní aplikaci. Když uživatel vyplní údaje v prvním kroku tvorby životopisu, je mu účet automaticky vytvořen. Refaktorovaná aplikace, na rozdíl od původní aplikace, dává uživateli možnost si nastavit heslo (viz obr. 26).



Obr. 26 Dialogové okno při vytváření nového uživatelského účtu.

Uživatel s existujícím účtem se může přihlásit v kterékoliv části aplikace pomocí tlačítka *Přihlásit se* v pravém horním rohu uživatelského rozhraní. To otevře dialogové okno přihlášení (viz obr. 27). V případě odhlášení, které lze provést tlačítkem *Odhlásit se*, jenž se nachází v menu s uživatelskými akcemi[33] je uživatel přesměrován na dedikovanou stránku přihlášení uživatele[34].



Obr. 27 Dialogové okno přihlášení.

Přihlášený uživatel má také přístup do uživatelské části aplikace, kde se nachází přehled vytvořených životopisů a nastavení[35] (viz obr. 28).

Obr. 28 Uživatelská část aplikace v rozlišení o šířce 1920px, 768px a 400px respektive.

Akce u každého životopisu jsou z leva doprava: úprava životopisu, zobrazení náhledu životopisu, stažení životopisu (přejde na příslušný krok) a smazání životopisu.

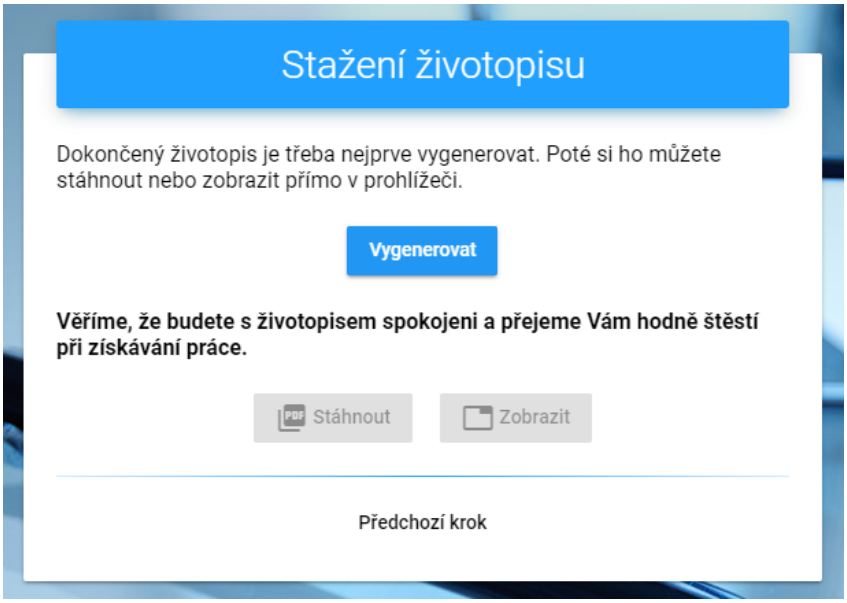
[33] Uživatelské menu se otevírá tlačítkem s názvem e-mailové adresy, jenž nahrazuje tlačítko *Přihlásit se.*

[34] Na tuto stránku je nepřihlášený uživatel přesměrován také pokud se snaží přímo zadáním URL dostat do části aplikace, kam nemá nepřihlášený uživatel přístup.

[35] V současnosti lze pouze změnit heslo.

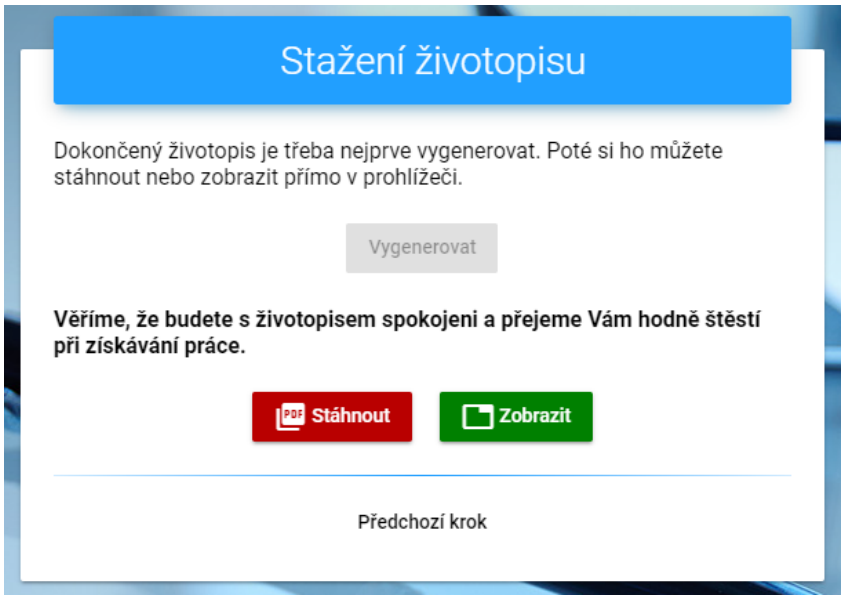
6.2.5 Stažení životopisu

V kroku Stažení životopisu může přihlášený uživatel svůj životopis kdykoliv vygenerovat (viz obr. 29).



Obr. 29 Generování životopisu.

Následně si může vygenerované PDF stáhnout nebo zobrazit v prohlížeči (viz obr. 30).



Obr. 30 Vygenerované PDF lze stáhnout nebo zobrazit v prohlížeči.

6.2.6 Zpracování zdrojových dat

S načítáním a ukládáním zdrojových dat výrazně pomáhá knihovna Angular 2 JSON API[36]. Podobně jako se u API vytvářela schémata pro knihovnu Neomerx JSON API (viz podkapitola 5.4), ve SPA se vytváří modely jednotlivých zdrojů pro knihovnu Angular2 JSON API. Př. modelu zdroje *Vzdělání*:

```
82  @JsonApiModelConfig({
83    type: 'educations'
84  })
85  export class Education extends JsonApiModel {
86    @BelongsTo()
87    curriculum: Curriculum;
88
89    @Attribute()
90    dateFrom: string;
91    @Attribute()
92    dateTo: string;
93    @Attribute()
94    specialization: string;
95    @Attribute()
96    qualification: string;
97    @Attribute()
98    organization: string;
99    @Attribute()
100   keyClasses: string;
101   @Attribute()
102   additionalInfo: string;
103 }
```

Řádek 82[37]-84: Definování typu zdroje.

ř. 85: Dědí se třída knihovny Angular 2 JSON API.

ř. 86 a 87: Definování vazby na životopis.

ř. 89-102: Definování atributů.

Vytvořený model se přiřadí v úložišti neboli třídě `<Datastore>`:

```
104 @Injectable()
105 @JsonApiDatastoreConfig({
106   baseUrl: 'https://www.example.com/api/',
107   models: {
108     curricula: Curriculum,
109     educations: Education
```

[36] Angular 2 JSON API je adaptér pro JSON API. (Ghidoli, 2016)

[37] @JsonApiModelConfig(), @BelongsTo(), @Attribute() apod. jsou tzv. dekorátory, viz https://angular-2-training-book.rangle.io/handout/features/decorators.html

```
110     }
111   })
112   export class ApiDatastore extends JsonApiDatastore {
113     constructor(http: Http) {
114       super(http);
115     }
116   }
```

Řádek 104: Třídu lze používat jako závislost.

ř.105-111: Nastavení úložiště. Nastavení URL adresy API (**106**) a načtení typů modelů (**107-110**).

ř. 113-115: Předání závislosti na službě *Http* (poskytované Angularem).

Následně lze již posílat požadavky použitím funkcí instance třídy `<Datastore>`:

```
117   datastore.findRecord(Education, '1').subscribe();
```

Řádek 117: Zašle požadavek GET cílící specifické *Vzdělání* s ID 1

Zdroje lze ukládat nebo měnit použitím funkce `<save>` přímo u instance modelu:

```
118   education.save().subscribe();
```

Řádek 118: Pošle požadavek POST na vytvoření nového *Vzdělání*. Pokud byl do instance modelu předtím načten již existující zdroj, pošle se požadavek PATCH se změnami.

6.3 Autentizace

V podkapitole 5.3 byla autentizace představena na straně API. Tato podkapitola se věnuje tomu, jak autentizace funguje na straně klientské aplikace. O všechny úkony spojené s autentizací uživatele se stará služba `<AuthService>` (ať už se jedná o přihlašování, odhlašování, obnovu tokenů nebo kontrolu, jestli je uživatel přihlášen).

6.3.1 Přihlášení uživatele

Přihlášení uživatele provádí funkce `<login>`.

```
119   login(): Promise<void> {
120     this.authenticating = true;
121     const json          = HttpHelper.getLoginJson(this.user);
122
123     return this.http.post(HttpHelper.loginUrl, json,
124   HttpHelper.getRequestOptions())
125       .toPromise()
126       .then((response: Response) => {
127         localStorage.setItem('email', this.user.email);
```

```
128         this.user.id = HttpHelper.extractTokens(response);
129         this.refresh();
130         this.snackBar.loginSnackBar();
131         this.authenticating = false;
132       })
133       .catch(error => {
134         if (error.status === 422) {
135           this.snackBar.wrongCredentialsSnackBar();
136         } else {
137           this.snackBar.errorSnackBar();
138         }
139         this.authenticating = false;
140
141         return Promise.reject(error);
142       });
143   }
```

Řádek 119: Funkce vrací *Promise*[38] (vše se odvíjí od úspěchu prováděného http požadavku).

ř. 120: Indikace, že začíná proces autentizace. Zobrazení indikace zpracovávání v uživatelském rozhraní.

ř. 121: Načtení obsahu[39] (formátovaného podle JSON API) požadavku.

ř. 123-125: Vrací se *Promise* (**125**) HTTP požadavku, který je zároveň odeslán (**123**). Předání nastavení (hlaviček) požadavku (**124**).

ř. 126-132: Požadavek úspěšně zpracován, vrácena odpověď (viz podkapitola 5.3.1) (**126**). Uložení e-mailové adresy do lokálního úložiště[40] (**127**). Načtení ID uživatele z odpovědi pomocí funkce pro zpracování tokenů (viz níže) (**128**). Spuštění obnovy tokenů (viz podkapitola 6.3.2) (**129**). Oznámení[41] o úspěšném přihlášení v uživatelském rozhraní (**130**). Konec indikace zpracovávání (**131**).

ř. 133-142: Požadavek nebyl zpracován, vrácena chyba (**133**). Pokud byly chybně zadané údaje (**134**), zobrazí se příslušné oznámení (**135**), jinak se zobrazí oznámení o technické chybě (**136**). Konec indikace zpracovávání (**139**). Vrací se výjimka, kterou může objekt volající tuto funkci odchytit a dále zpracovat (**141**).

Funkce `<extractTokens>`, pomocné třídy `<HttpHelper>`, využívá knihovnu Auth0 angular2-jwt[42], kterou je dekódován přístupový token obsažený v odpovědi.

---

[38] *Promise* reprezentuje eventuální výsledek asynchronní operace. (PromisesAplus.com, 2014)

[39] Přihlášení není zdrojový požadavek, takže nelze použít `<Datastore>`.

[40] Uložení dat v lokálním úložišti prohlížeče. (w3schools.com, 2017)

[41] Oznámení jsou zobrazována pomocí komponenty (Snack Bar) z knihovny Angular Material.

[42] Auth0 angular2-jwt je pomocná knihovna pro práci s JWT v Angular 2 aplikacích. (Auth0.com, 2016)

```
144  static extractTokens(response: Response): string {
145    const data = response.json().data;
146
147    const accessToken  = data['attributes']['accessToken'];
148    const refreshToken = data['attributes']['refreshToken'];
149
150    const jwtHelper    = new JwtHelper();
151    const decodedToken = jwtHelper.decodeToken(accessToken);
152    const userId       = decodedToken.uid;
153
154    localStorage.setItem('token', accessToken);
155    localStorage.setItem('refreshToken', refreshToken);
156    localStorage.setItem('userId', userId);
157
158    return userId;
159  }
```

Řádek 145: Načtení dat z odpovědi.

ř. 147 a 148: Načtení přístupového (**147**) a obnovovacího (**148**) tokenu z atributů dat.

ř. 150-152: Vytvoření objektu knihovny Auth0 angular2-jwt (**150**). Dekódování přístupového tokenu pomocí knihovny (**151**). Načtení ID uživatele z tokenu (**152**).

ř. 154-156: Uložení přístupového tokenu (**154**), obnovovacího tokenu (**155**) a ID uživatele (**156**) do lokálního úložiště.

ř. 158: Vrácení ID uživatele jako řetězec.

Pomocí knihovny Auth0 angular2-jwt, se také kontroluje, zdali je uživatel přihlášen.

```
160  get authenticated(): boolean {
161    return tokenNotExpired();
162  };
```

Řádek 161: Funkce `<tokenNotExpired>` z knihovny Auth0 angular2-jwt validuje přístupový token uložený v lokálním úložišti.

6.3.2 Obnova tokenů

Automatická obnova tokenů se spouští při úspěšném přihlášení (viz výše **ř. 129**).

```
163  refresh() {
164    const interval           = HttpHelper.refreshInterval;
165    this.refreshSubscription = Observable.interval(interval)
166      .flatMap(() => this.http.post(HttpHelper.refreshUrl,
167  HttpHelper.getAuthenticationJson(), HttpHelper.getRequestOptions())
168        .catch(() => {
169          this.router.navigate(['/user/login']);
```

```
170           this.snackBar.errorSnackBar();
171           return Observable.throw('Error');
172         }))
173       .subscribe((response: Response) => {
174         HttpHelper.extractTokens(response);
175       });
176   }
```

Řádek 164: Použití funkce `<refreshInterval>`[43] pomocné třídy `<HttpHelper>` pro načtení intervalu obnovy.

ř. 165 a 166: Vytvoření subskripce nad *Observable*[44] (**165**), které v pravidelných intervalech (o načtené délce) posílá požadavek na obnovu tokenu (**166**).

ř. 167: Předání nastavení požadavku.

ř. 168-172: Odchycení chybné odpovědi (**168**). Přesměrování na stránku s přihlášením (**169**). Přerušení odesílání požadavků (**171**).

ř. 173-174: Odchytávání odpovědí správně zpracovaných požadavků (**173**). Následné zpracování tokenů v odpovědi (**174**).

Proces automatické obnovy tokenů se spouští také při každém spuštění aplikace[45]. V tomto případě se ještě také kontroluje čas uloženého přístupového tokenu a požadavek se pošle až těsně před jeho expirací[46].

6.3.3 Odhlášení uživatele

Odhlášení uživatele provádí funkce `<logout>`.

```
177   logout(): Promise<void | never> {
178     const json = HttpHelper.getAuthenticationJson();
179
180     return this.http.post(HttpHelper.logoutUrl, json,
181   HttpHelper.getRequestOptions(true))
182       .toPromise()
183       .then(() => {
184         this.user = this.dataStore.createRecord(User);
185         localStorage.clear();
186         this.stopRefresh();
187         this.snackBar.logoutSnackBar();
```

[43] Funkce načte z uloženého přístupového tokenu délku intervalu odpovídající rozdílu mezi časem vytvoření a časem expirace tokenu. Pro vytažení potřebných informací z tokenu se opět používá knihovna *Auth0 angular2-jwt.*

[44] *Observable* je objekt knihovny *ReactiveX*, jenž vysílá změny hodnot. Tyto změny pozorovatelské objekty odchytávají a případně samy iniciují. (ReactiveX.io, 2017)

[45] Pokud je uložen validní (neprošlý) obnovovací token.

[46] Jestliže přístupový token již není platný, požadavek na obnovení se pošle okamžitě.

```
188       })
189       .catch(() => {
190         this.snackBar.errorSnackBar();
191       });
192   }
```

Řádek 182: Předání nastavení[47] požadavku.

ř. 183-188: Při úspěšném požadavku se vytvoří nový (prázdný) model uživatele (**184**). Vymažou se všechna data uložená v lokálním úložišti (**185**). Zastaví se obnova tokenů (**186**) a v uživatelském rozhraní se zobrazí příslušné oznámení o odhlášení (**187**).

6.4 Generování PDF

Poslední z funkcí klientské aplikace, jejíž druhá polovina zpracování se provádí v rámci API (viz podkapitola 5.5).

```
193   createCurriculumPdf(curriculumId: string) {
194     this.generating = true;
195     const json      = HttpHelper.getPdfRequestJson(curriculumId);
196
197     this.http.post(HttpHelper.pdfUrl, json,
198   HttpHelper.getRequestOptions(true))
199       .subscribe(
200         response => {
201           const encodedString = response.json().data.attributes.file;
202           const decodedString = atob(encodedString);
203
204           const arrayBuffer = new ArrayBuffer(decodedString.length);
205           const intArray    = new Uint8Array(arrayBuffer);
206           for (let i = 0; i < decodedString.length; i++) {
207             intArray[i] = decodedString.charCodeAt(i);
208           }
209
210           this.pdfFile = new Blob([intArray], { type:
211   'application/pdf' });
212           const url    = window.URL.createObjectURL(this.pdfFile);
213           this.pdfLink =
214   this.sanitizer.bypassSecurityTrustResourceUrl(url);
215           this.generating = false;
216         },
217         error => {
218           this.snackBar.errorSnackBar();
```

[47] Hodnota *true* předaná v parametru funkce `<getRequestOptions>` indikuje, že se má přidat autorizační hlavička.

```
219           }
220         );
221     }
```

Řádek 201 a 202: Načtení PDF jako řetězec zakódovaný v base64 (**201**). Dekódování řetězce (**202**).

ř. 204-208: Načtení řetězce do pole bytů.

ř. 210 a 211: Vytvoření PDF BLOB[48] objektu.

ř. 212-214: Vytvoření URL pro otevření PDF objektu (**212**) a nastavení toho že je URL důvěryhodná[49] (**213** a **214**).

Možnost uživatele stáhnout životopis je implementována pomocí knihovny FileSaver[50].

```
222   FileSaver.saveAs(this.pdfService.pdfFile, 'CV-' + fullName.replace('
223', '_'));
```

Řádek 222 a 223: Nabídnutí stažení PDF souboru. Prvním předaným parametrem je PDF BLOB, druhým jméno souboru jako řetězec.

[48] Binary large object

[49] Angular jinak zablokuje otevření URL.

[50] https://github.com/eligrey/FileSaver.js/

7 Nasazení do produkce

Ostrému nasazení refaktorované aplikace do produkčního prostředí bude předcházet spuštění beta aplikace, jenž se nachází na subdoméně beta.curriculum-generator.com.

7.1 Beta verze

Pro beta verzi aplikace vytvořím novou databázi[51] obsahující pouze tabulky nového databázového modelu. Po vytvoření databáze nahraji API na subdoménu api.curriculum-generator.com a vyzkouším její dostupnost (jednoduchým požadavkem). Následně již nahraji nově vytvořenou single-page aplikaci a otestuji kompletní funkčnost celé aplikace. Dalším krokem bude importování dat do databáze beta verze z databáze původní a následující zajištění funkčnosti aplikace i v tomto stadiu.

Jakmile bude aplikace funkční, prostřednictvím sociálních medií budou informování uživatelé, kterým se beta verze aplikace volně zpřístupní. Odladí se případné chyby, na které uživatelé při používání aplikace narazí a jenž se při interním testování aplikace nepodařilo odchytit. Mezitím bude do aplikace integrována platební brána. Aplikace by poté měla být již plně připravena pro nasazení do produkce

7.2 Produkce

V rámci nasazení do produkce se, při dostatečné konzistenci dat, naimportuje databáze beta verze také pro produkci. A to včetně dat přidaných v rámci beta verze. Tento stav databáze bude výchozím bodem pro další provoz aplikace a měla by tak být zachována všechna data, jak z původní, tak i z beta verze aplikace. Provoz API bude zachován na stávající subdoméně api.curriculum-generator.com.

Posledním krokem tak bude samotné nahrání finální, druhé produkční verze aplikace Curriculum Generator na doménu www.curriculum-generator.com, čímž bude celý proces refaktorování dokončen.

[51] Prostřednictvím migračních souborů vytvořených při implementaci aplikace.

8 Závěr

V této práci byl popsán proces refaktorování 6 let staré webové aplikace pro tvorbu životopisů. Podle analýzy původní aplikace vytvořené pomocí PHP frameworku Nette, byl vytvořen návrh nové aplikace, složené ze dvou samostatných částí, serverové části v podobě API a klientské části v podobě single-page aplikace. V rámci návrhu také byla zjednodušena databáze, která má nyní přibližně polovinu tabulek oproti databázi původní. Následně bylo API implementováno pomocí PHP frameworku Lumen a single-page aplikace implementována pomocí JavaScriptového frameworku Angular.

Refaktorovaná aplikace obsahuje řadu změn a několik nových rozšíření, které ji činí uživatelsky přívětivější oproti aplikaci původní. Nemusí se při vykonání HTTP požadavku znovu načítat, díky čemuž pohotověji reaguje na akce uživatele. Část aplikace na tvorbu životopisu byla rozšířena o interaktivní náhled životopisu, který okamžitě reaguje na prováděné změny a reprezentuje podobu výsledného životopisu. Některé části aplikace nově obsahují boční menu, ulehčující navigaci v rámci aplikace. Uživatel se tak dostane z kteréhokoliv místa v aplikaci na kterékoliv jiné místo aplikace nejvýše 4 akcemi. Aplikace byla rozšířena o uživatelskou část, ve které lze měnit heslo a spravovat životopisy. Těch nyní může mít jeden uživatelský účet současně několik. V neposlední řadě byla aplikace optimalizována pro použití na mobilních zařízeních. Výsledná aplikace se v současnosti nachází na webové adrese beta.curriculum-generator.com.

9 Literatura

ANGULAR. *Angular releases* [online]. 2016, ©2017 [cit. 2017-03-15]. Dostupné z: https://github.com/angular/angular/releases/tag/2.0.0

ANGULAR. *Forms* [online]. ©2010-2017 [cit. 2017-05-07]. Dostupné z: https://angular.io/docs/ts/latest/guide/forms.html

ANGULAR. *Master/Detail* [online]. ©2010-2017 [cit. 2017-03-15]. Dostupné z: https://angular.io/docs/ts/latest/tutorial/toh-pt2.html

ANGULAR. *One Framework. Mobile & desktop.* [online]. ©2010-2017 [cit. 2017-03-15]. Dostupné z: https://angular.io/

ANGULAR. *Tutorial: Tour of heroes* [online]. ©2010-2017 [cit. 2017-03-15]. Dostupné z: https://angular.io/docs/ts/latest/tutorial/

ANGULARJS. *AngularJS releases* [online]. 2010 ©2017 [cit. 2017-05-07]. Dostupné z: https://github.com/angular/angular.js/releases?after=v0.9.9

ANGULARJS. *Forms* [online]. ©2010-2017 [cit. 2017-05-07]. Dostupné z: https://docs.angularjs.org/guide/forms

ANGULARJS. *PhoneCat Tutorial App* [online]. ©2010-2017 [cit. 2017-05-07]. Dostupné z: https://docs.angularjs.org/tutorial

ANGULARJS. *What Is AngularJS?* [online]. ©2010-2017 [cit. 2017-03-11]. Dostupné z: https://docs.angularjs.org/guide/introduction

ANGULAR MATERIAL. *Component Library* [online]. ©2010-2017 [cit. 2017-05-17]. Dostupné z: https://material.angular.io/components

ANGULAR MATERIAL. *Material Design for Angular* [online]. 2016 ©2017 [cit. 2017-05-17]. Dostupné z: https://github.com/angular/material2

AUTH0. *angular2-jwt* [online]. 2016 ©2017 [cit. 2017-05-16]. Dostupné z: https://github.com/auth0/angular2-jwt

COBUCCI, L. *JWT* [online]. 2014 ©2017 [cit. 2017-05-17]. Dostupné z: https://github.com/lcobucci/jwt

CURRICULUM GENERATOR. *Curriculum Generator* [online]. ©2011-2016 [cit. 2017-05-05]. Dostupné z: http://www.curriculum-generator.com

EMBER. *A framework for creating ambitious web applications.* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: http://emberjs.com

EMBER. *Creating Your App* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: https://guides.emberjs.com/v2.13.0/tutorial/ember-cli/

EMBER. *Input Helpers.* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: https://guides.emberjs.com/v2.13.0/templates/input-helpers/

EMBER. *Previous releases* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: http://emberjs.com/builds/tagged

EMBER. *The team behind ember* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: http://emberjs.com/team/

EMBERCLI. *The command line interface for ambitious web applications* [online]. 2017 [cit. 2017-03-18]. Dostupné z: https://ember-cli.com/

FACEBOOK. *React* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: https://facebook.github.io/react/

FACEBOOK. *Tutorial: Intro To React* [online]. ©2017 [cit. 2017-03-18]. Dostupné z: https://facebook.github.io/react/tutorial/tutorial.html

FIELDING, T. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. Dostupné z: http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf

GHIDOLI, D. *Angular2 JSON API* [online]. 2016 [cit. 2017-05-16]. Dostupné z: https://github.com/ghidoz/angular2-jsonapi

GITHUB. *JavaScript framework* [online]. ©2017 [cit. 2017-03-19]. Dostupné z: https://github.com/search?o=desc&q=JavaScript+framework&s=stars&type=Repositories&utf8=%E2%9C%93

GLOBAL STATS: STAT COUNTER. *Screen Resolution Stats Worldwide* [online]. ©1999-2017 [cit. 2017-05-05]. Dostupné z: http://gs.statcounter.com/screen-resolution-stats

HABIB, O. *PHP Microframework vs. Full Stack Framework*. In: AppDynamics [online]. 2015 ©2009-2017 [cit. 2017-05-14]. Dostupné z: https://blog.appdynamics.com/engineering/php-microframework-vs-full-stack-framework/

HANDLEBARS. *Handlebars* [online]. 2017 [cit. 2017-03-18]. Dostupné z: http://handlebarsjs.com/

IINTERNET ARCHIVE. *Wayback Machine* [online]. ©2017 [cit. 2017-05-05]. Dostupné z: https://archive.org/web/

JSON API. *JSON API* [online]. 2015, ©2017 [cit. 2017-05-08]. Dostupné z: https://github.com/json-api/json-api

JSON API. *Specification: Errors* [online]. 2015 ©2017 [cit. 2017-05-08]. Dostupné z: http://jsonapi.org/format/#errors

KEEFER, C. *The What and Why of Javascript Frameworks*. In: ART & LOGIC [online]. 2015, ©2017 [cit. 2017-03-11]. Dostupné z: https://artandlogic.com/2015/05/the-what-and-why-of-javascript-frameworks/

LUMEN. *Lumen.* [online]. ©2017 [cit. 2017-05-17]. Dostupné z: https://lumen.laravel.com/

MICROSOFT. *JavaScript that scales* [online]. ©2012-2016 [cit. 2017-03-15]. Dostupné z: http://www.typescriptlang.org/

MIKOWSKI, M., POWELL, J. *Single page web applications: JavaScript end-to-end*. Manning Publications, 2012. ISBN 9781617290756.

MPDF. *MPDF [online].* ©2015 [cit. 2017-05-05]. Dostupné z: https://mpdf.github.io/

NEOMERX. *Framework agnostic JSON API implementation* [online]. 2017 [cit. 2017-05-16]. Dostupné z: https://github.com/neomerx/json-api

NETTE. *Nette* [online]. ©2008-2017 [cit. 2017-05-05]. Dostupné z: https://nette.org/cs/

NETTE. *Nette releases* [online]. 2012 [cit. 2017-05-05]. Dostupné z: https://github.com/nette/nette/releases/tag/v2.0.0

PROMISES/A+. *Promises/A+* [online]. 2014 [cit. 2017-05-16]. Dostupné z: https://promisesaplus.com/

RAVENSBORG-GJERTSEN, N. *Laravel PDF: mPDF wrapper for Laravel 5* [online]. 2016 [cit. 2017-05-16]. Dostupné z: https://github.com/niklasravnsborg/laravel-pdf

React (JavaScript library). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-18]. Dostupné z: https://en.wikipedia.org/wiki/React_(JavaScript_library)

REACTIVEX. *Observable* [online]. ©2017 [cit. 2017-05-16]. Dostupné z: http://reactivex.io/documentation/observable.html

STOWE, M. *Undisturbed REST: A Guide to Designing the Perfect API*. San Francisco: MuleSoft, 2015. ISBN 978-1-329-11656-6.

VUE.JS. *Comparison with Other Frameworks* [online]. ©2014-2017 [cit. 2017-03-19]. Dostupné z: https://vuejs.org/v2/guide/comparison.html

VUE.JS. *Introduction* [online]. ©2014-2017 [cit. 2017-03-19]. Dostupné z: https://vuejs.org/v2/guide/

VUE.JS. *The Progressive JavaScript Framework* [online]. ©2014-2017 [cit. 2017-03-19]. Dostupné z: https://vuejs.org/

VUE.JS. *Vue releases* [online]. 2013 [cit. 2017-03-19]. Dostupné z: https://github.com/vuejs/vue/tags?after=v0.7.5

W3SCHOOLS.COM. *HTML5 Local Storage* [online]. ©1999-2017 [cit. 2017-05-16]. Dostupné z: https://www.w3schools.com/html/html5_webstorage.asp

Přílohy

A Obsah CD

- Schéma databázového modelu aplikace Curriculum Generator
- Schéma databázového modelu refaktorované aplikace
- Soubor databázového systému
- Obrázky uživatelského rozhraní refaktorované aplikace
- Zdrojový kód REST API
- Zdrojový kód single-page aplikace
- Dokumentace REST API